

# Einführung in python™

Ausgabe 10.2022

Copyright © office@iten-engineering.ch  
Alle Rechte vorbehalten.  
Reproduktion (auch auszugsweise) ist nur mit schriftlicher Bewilligung des Verfassers gestattet.

1

## Inhalt

Kapitel	Inhalt	Folie
1	Kick Start (Einführung, Hello World)	3
2	Grundlagen	11
3	Datenstrukturen (Strings, List, Tuples, Sets, Dictionaries)	52
4	Klassen & Objekte	89
5	File Input/Output	101
6	Module & Packages	108
7	Standard Libraries (math, os, sys, subprocess, re)	123
8	Data Science Libraries (NumPy, Pandas, SciPy, SciKit, Mathplot)	150
9	Anwendungsbeispiele	184
10	Weitere Übungen	186
11	Literatur & Weblinks	188
12	Anhang	190

2

## Kapitel 1

### Kick Start

3

### Was ist Python?

- ▶ Python is an interpreted, interactive, object-oriented programming language.
- ▶ It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes.
- ▶ It supports multiple programming paradigms beyond object-oriented programming, such as procedural and functional programming.
- ▶ Python combines remarkable power with very clear syntax.

Quelle: <https://docs.python.org/3/faq/general.html>

4

## Was ist Python?

- ▶ It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++.
- ▶ It is also usable as an extension language for applications that need a programmable interface.
- ▶ Finally, Python is portable: it runs on many Unix variants including Linux and macOS, and on Windows.

Quelle: <https://docs.python.org/3/faq/general.html>

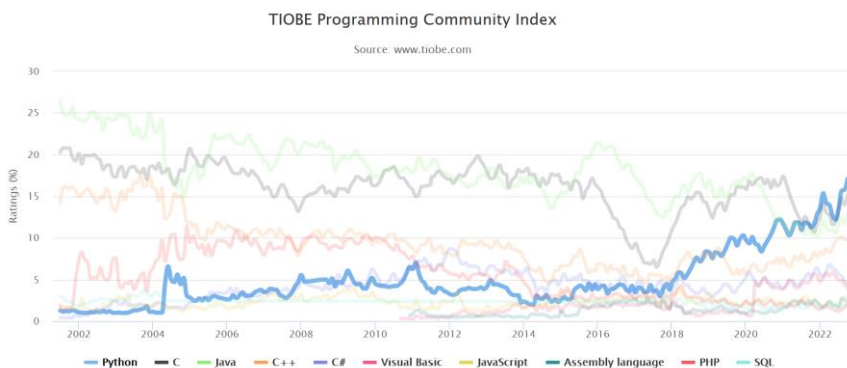
## Kompiliert vs. Interpretiert

- |  |  |
|--|--|
| ▶ Source Code wird vor Ausführung kompiliert                               | ▶ Source Code wird während der Laufzeit interpretiert und ausgeführt |
| ▶ Erzeugt ausführbare Datei für spezifische Architektur und Betriebssystem | ▶ Ausführung ist langsamer   |
| ▶ Ausführung des Programm ist schnell                                      | ▶ Interaktives Programmieren ist möglich                             |

## Warum Python?

- ▶ Interpretierte Sprachen sind ideal für interactive Arbeiten, Forschung, etc.
- ▶ Einfache Syntax
- ▶ Mächtige Bibliotheken
- ▶ Sehr populär im Bereich Data Science (Artificial Intelligence, Machine Learning)
- ▶ Interaktive Arbeitsweise mit Jupiter Notebooks
- ▶ Mächtige Grafik Bibilotheken
- ▶ Frei verfügbar (Open Source)
- ▶ Erweiterbar und Einbindbar (C, C++)

## Warum Python?



Quelle: <https://www.tiobe.com/tiobe-index>

## Demo

► hello.py

```
print("Hello World")
```

► hello.ipynb

**Jupyter Notebook**

- Ist eine Mischung aus Text (Markdown) und ausführbarem Code
- Die Code Zellen können einzeln oder auch alle zusammen ausgeführt werden.
- Die Resultate werden dann unterhalb der jeweiligen Code Zelle eingeblendet.

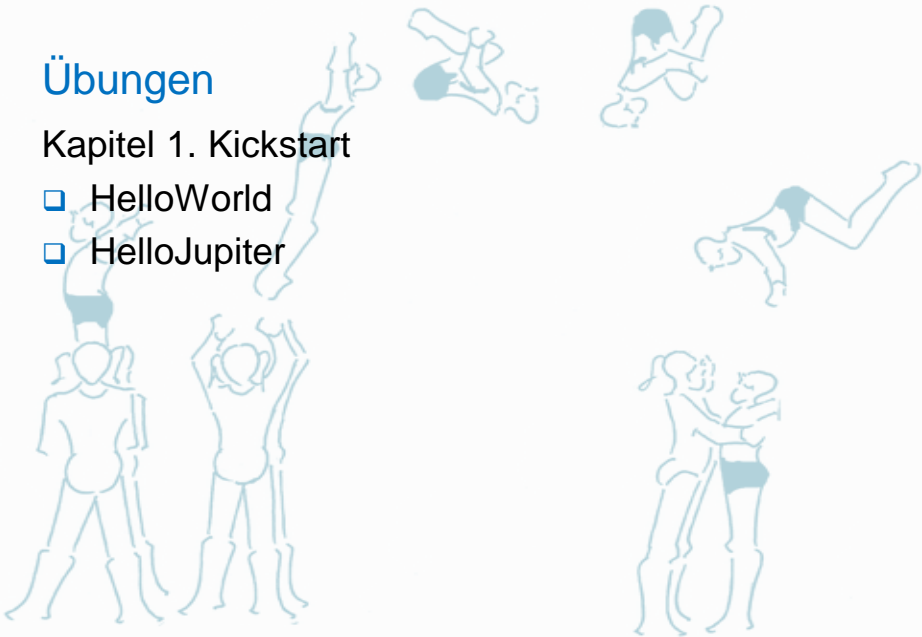
Jupyter Notebook sind ideal für interaktive Arbeiten die gleichzeitig dokumentiert werden sollen oder Grafiken darstellen.

```
1 print("Hello Jupiter")  
Hello Jupiter  
  
2 4 + 4  
2 16  
  
The end.
```

## Übungen

### Kapitel 1. Kickstart

- ☐ HelloWorld
- ☐ HelloJupiter



## Kapitel 2

# Grundlagen

11

## Abschnitt I

# Operatoren, Variablen, Kommentare, Hilfe & Typ Informationen

12

## Operatoren

Python unterstützt folgende Typen von Operatoren:

- ▶ Arithmetic Operators
- ▶ Comparison (Relational) Operators
- ▶ Assignment Operators
- ▶ Logical Operators
- ▶ Bitwise Operators
- ▶ Membership Operators
- ▶ Identity Operators

## Operatoren II

Operator	Bezeichnung	Beispiel	
+x, -x	Vorzeichen	-3	
+, -	Addition, Subtraktion	10 - 3	Ergebnis: 7
*, /, %	Multiplikation, Division, Rest	27 % 7	Ergebnis: 6
//	Ganzzahl Division	7//2	Ergebnis: 3
**	Potenz	10 ** 3	Ergebnis: 1000
+=, -=, *=, /=, **=	Compound Operatoren	x += 3	Ergebnis: x = x + 3
or, and, not	Boolsches OR, AND, NOT	(a or b) and c	
in	Element in Menge	1 in [0, 1]	Ergebnis: True
<, <=, >, >=, !=, ==	Vergleichsoperatoren	2 <= 3	Ergebnis: False
is, is not	Vergleich Objekte	x=5; y=x; x is y	Ergebnis: True
~, &, ^, ~x	Bitweises OR, AND, XOR, NOT	6 ^ 3	Ergebnis: 4
<<, >>	Shiftoperatoren	6 << 2	Ergebnis: 24

Details: [https://docs.python.org/3/reference/lexical\\_analysis.html#operators](https://docs.python.org/3/reference/lexical_analysis.html#operators)

## Operator Precedence

Operator	Description
<code>=</code>	Assignment expression
<code>lambda</code>	Lambda expression
<code>if - else</code>	Conditional expression
<code>or</code>	Boolean OR
<code>and</code>	Boolean AND
<code>not x</code>	Boolean NOT
<code>in, not in, is, is not, &lt;, &lt;=, &gt;, &gt;=, !=, ==</code>	Comparisons, including membership tests and identity tests
<code> </code>	Bitwise OR
<code>^</code>	Bitwise XOR
<code>&amp;</code>	Bitwise AND
<code>&lt;&lt;, &gt;&gt;</code>	Shifts
<code>+, -</code>	Addition and subtraction
<code>*, @, /, //, %</code>	Multiplication, matrix multiplication, division, floor division, remainder [5]
<code>+, -, ~, ~x</code>	Positive, negative, bitwise NOT
<code>**</code>	Exponentiation [6]
<code>await x</code>	Await expression
<code>x[index], x[index:index], x(arguments...), x.attribute</code>	Subscription, slicing, call, attribute reference
<code>(expressions...), [expressions...], {key: value...}, {expressions...}</code>	Binding or parenthesized expression, list display, dictionary display, set display

Quelle:  
<https://docs.python.org/3/reference/expressions.html?highlight=precedence>

15

## Variablen

```
counter = 100           # Integer
miles   = 1000.0        # Floating point
name    = "John"        # String (instanciert vom Literal "John")
age     = str(54)        # String (instanciert via Konstruktor)

print("counter = ", counter)
print("miles   = ", miles)
print("name    = ", name)
print("age     = ", age)
```

```
counter = 100
miles   = 1000.0
name    = John
Age     = 54
```

Output

16



## Statische vs. dynamische Typdeklaration

- ▶ Sprachen wie C oder Java haben eine statische Typdeklaration
  - Dass heisst der Typ einer Variablen kann während der Laufzeit nicht ändern
  - Nur der Wert der Variablen kann sich ändern
- ▶ Python ordnet den Typ einer Variablen dynamisch zu
  - Es erfolgt keine Typangabe bei der Deklaration
  - Es kann sowohl der Wert als auch der Typ einer Variablen zur Laufzeit ändern

17

## Dynamische Typenzuordnung

```
i = 42
print(type(i))

i = "Hallo"
print(type(i))

i = [3, 9, 17]
print(type(i))
```

```
<class 'int'>
<class 'str'>
<class 'list'>
```

Output

18

## Datentypen

Typ	Klasse
Text	str
Numeric	int, float, complex
Sequence	list, tuple, range
Map	dict
Set	set, frozenset
Boolean	bool
Binary	bytes, bytearray, memoryview

```
solution = 42
print(type(solution))
print(isinstance(solution, int))
```

Datentyp anzeigen & prüfen

```
<class 'int'>
True
```

Output

## Casting

```
s = "12.5"
print(type(s))
print(s)

f = float(s)
print(type(f))
print(f)

i = int(s)
```

```
<class 'str'>
12.5
<class 'float'>
12.5
Traceback (most recent call last):
  File ".../02-basics/casting.py", line 9, in <module> i = int(s)
ValueError: invalid literal for int() with base 10: '12.5'
```

Output

## Variablen Namen

- ▶ Müssen mit **Buchstabe** oder **Unterstrich** „\_“ beginnen.
- ▶ Die weiteren Zeichen dürfen sich aus einer beliebigen Folge von **Buchstaben**, **Ziffern** und dem **Unterstrich** zusammensetzen.
- ▶ Variablennamen sind **case-sensitive**:
  - Das bedeutet, dass Python zwischen Gross- und Kleinschreibung unterscheidet
- ▶ **Keine reservierten** Worte

## Reservierte Wörter

- ▶ if, elif, else
- ▶ True, False, None
- ▶ and, or, not
- ▶ for, while, continue, break
- ▶ try, except, finally, raise
- ▶ def, class, lambda
- ▶ return, yield, pass
- ▶ from, import, as, with
- ▶ is, in, assert
- ▶ del, global, nonlocal

## Kommentare

```
def test_elapsed(self):
    # init
    sw = Stopwatch()
    # test
    sw.start()
    time.sleep(3)
    sw.stop()
    # debug
    if (self.debug):
        print("Test Stopwatch:")
        print("Start ", sw.start_time)
        print("Stop  ", sw.stop_time)
        print("Elapsed", sw.elapsed())
    # check
    self.assertTrue(sw.stop_counter - sw.start_counter >= 2.9) # give some tolerance
```

Bis zum Ende  
der Zeile

## Kommentare II

```
class ReportSystemPerformance(AbstractReport):
    """Report the system performance of the models.

    Sample Output
    -----
    ReportSystemPerformance
    Time CPU Memory Peak
    LangDetect 4.31 8.33 499253
    LangDetectSpacy 12.83 8.33 1336925
    LangFromStopwords 0.10 8.31 495957
    LangFromChars 65.28 17.18 282294433
    AzureTextAnalytics 146.70 1.18 594964
    Time elapsed: 700.58 sec

    Report saved to: outcome/ReportSystemPerformance.csv
    """

    def __init__(self):
        AbstractReport.__init__(self, "ReportSystemPerformance")
        self.model = None
        self.csv_file = "articles_all_1k.csv"
```

Mehrere  
Zeilen

## Kommentare III

```
def predict(self, text):
    """Predict the language code for the given text.
    Args:
        text (str): The text to predict the language of.
    Returns:
        str: The language code (ISO-639-1)
    """
    raise NotImplementedError("The method is not implemented yet.")
```

Beispiel  
Methode

25

## Hilfe und Type Information

```
obj = str
help(obj)          # show help of this object / function
print(obj)         # print object to default output
type(obj)          # show type of object

dir()              # List all variables of the namespace
dir(obj)           # List all attributes of object instance

c = complex        # class
c.__dict__         # show attributes of class
vars(c)            # show attributes of class

class Foo(object): # show instance attributes
    def __init__(self):
        self.a = 1
        self.b = 2

vars(Foo())         #==> {'a': 1, 'b': 2}
vars(Foo()).keys()  #==> ['a', 'b']
```

26

# Übungen

## Kapitel 2.1 Operatoren & Variablen

- ☐ Rechteck
- ☐ Fahrenheit



The background of the slide features a light blue illustration of several stylized human figures in various acrobatic poses, including handstands, backflips, and group formations, set against a white background.

---

Copyright © iten-engineering.ch Python Einführung 27

27

## Abschnitt II

Kontrollstrukturen

---

Copyright © iten-engineering.ch Python Einführung 28

28

## if – elif – else

```
seq = [1,2,3]

if len(seq) == 0:
    print("sequence is empty")
elif len(seq) == 1:
    print("sequence contains one element")
else:
    print("sequence contains several elements")
```

```
sequence contains several elements
```

Output

Beachte:

- ▶ Code Blocks werden durch **Einrückungen** definiert
- ▶ Es werden **keine Klammern** verwendet!

## if shorthand (Ternary Operator)

```
a = 5
b = 3

x = 10 if a > b else 1 # better readable

y = a > b and 10 or 1 # style more like java or other languages

print(x)
print(y)
```

```
10
10
```

Output

## for loop

```
# For loop over list of words  
words = ["winter", "is", "coming"]  
for word in words:  
    print(word)
```

```
winter  
is  
coming
```

Output

For Loop steuern:

- ▶ Mit **break** wird der for loop verlassen
- ▶ Mit **continue** wird die nächste Iteration des for loop ausgeführt

## for loop in reverse order

```
# For loop in reverse order  
words = ["coming", "is", "winter"]  
for word in reversed(words):  
    print(word)
```

```
winter  
is  
coming
```

Output



## for loop over multiple arrays with zip

```
# For loop over mutiple arrays (zip)
names = ["Peter", "Jane", "Fred"]
ages = [31, 35, 4]

for t in zip(names, ages):           # use a tuple
    print(t)

for name, age in zip(names, ages):  # use multiple loop-variables
    print(name, "is", age)

for i, name in enumerate(names):    # use enumerate and index
    print(name, "is", ages[i])

for i in range(len(names)):         # use range and index
    print(names[i], "is", ages[i])
```

33

## for loop with range and enumerate

```
# For loop with range
for i in range(10):           # i = 0..9
    print(i)

for i in range(5,10):         # i = 5..9
    print(i)

for i in range(0,100,10):     # i = 0, 10, 20, ... 80, 90
    print(i)

# For loop with index and element (enumerate)
list = ["first", "second"]
for pair in enumerate(list):
    print(type(pair), pair)    # <class 'tuple'> (0, 'first')
                                # <class 'tuple'> (1, 'second')
```

Range (start, end, step)  
 - Startwert (inklusive),  
 - Endwert (exklusiv),

34

## List Comprehension

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
```

```
for x in fruits:
    if "a" in x:
        newlist.append(x)
```

Konventioneller for Loop  
mit Erstellung neuer Liste  
für die Elemente welche  
ein "a" beinhalten

*# List Comprehension*

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
```

```
newlist = [x for x in fruits if "a" in x]
```

Gleiches Resultat mit  
List Comprehension

- List Comprehension ermöglicht den Einsatz einer kompakten Syntax (Anstelle eines for Loop) für die Erstellung einer neuen Liste basierend auf Werten einer bestehenden Liste.

## while

```
data = [1, 5, 4, 3, 4, 1, 8]
i = 0
```

```
sum = 0
count = 0
```

```
while (i < len(data)):
    value = data[i]
    sum = sum + value
    count = count + 1
    i = i+1;
```

```
print(data)
print(sum)
```

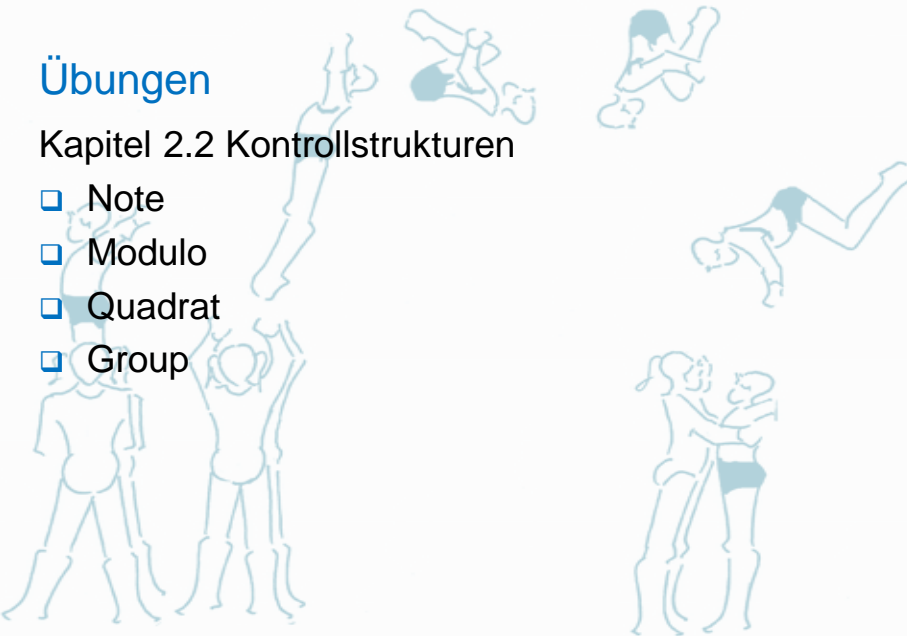
```
[1, 5, 4, 3, 4, 1, 8]
26
```

Output

# Übungen

## Kapitel 2.2 Kontrollstrukturen

- ☐ Note
- ☐ Modulo
- ☐ Quadrat
- ☐ Group



Copyright © iten-engineering.ch Python Einführung 37

37

## Abschnitt III

# Fehlerhandling mit Exception

Copyright © iten-engineering.ch Python Einführung 38

38

## Fehlerhandling mit Exception

- ▶ Bei einem Fehler (Exception)
  - stoppt Python die Ausführung normalerweise
  - Und gibt eine Fehlermeldung aus
- ▶ Möchte man die Fehler im Code behandeln, macht man dies mit:
  - `try` Start von Anweisungen die eine Exception auslösen können
  - `except` “Fangen” der Exception und Anweisungen für die Fehlerbehandlung
  - `finally` Abschluss Block mit Anweisungen die sowohl im Gut- als auch Fehlerfall ausgeführt werden

39

## try – except

```
try:
    input = "12x"
    nr = int(input)
except:
    print("Invalid input: ", input)
```

```
try:
    input = "12x"
    nr = int(input)
except Exception as e:
    print("Invalid input: ", str(e))
```

```
Invalid input: 12x
Invalid input: invalid literal for int() with base 10: '12x'
```

Output

40

## try – except II

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except (RuntimeError, TypeError, NameError):
    print("Catch all other expected errors.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

Spezifische Fehlerbehandlung

Mehrere Fehler fangen bei gleicher Fehlerbehandlung

Alle restlichen Fehler behandeln und mit raise weiter propagieren!

Output

```
OS error: [Errno 2] No such file or directory: 'myfile.txt'
```

Details: <https://docs.python.org/3/tutorial/errors.html>

41

## try – except – else – finally

```
def divide(x, y):
    success = True
    try:
        result = x / y
    except ZeroDivisionError:
        success = False
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        if (success):
            print("> division successfully done")
        else:
            print("> division failed")

divide(10,2)
divide(10,0)
divide(10,5)
```

Wird nur ausgeführt falls try Block ohne Fehler

Wird immer ausgeführt

Output

```
result is 5.0
> division successfully done
division by zero!
> division failed
result is 2.0
> division successfully done
```

42

## raise

```
try:
    input = -1
    if input < 0:
        raise Exception("Number must be positive")
except Exception as e:
    print("Invalid input:", str(e))
```

Invalid input: Number must be positive

Output

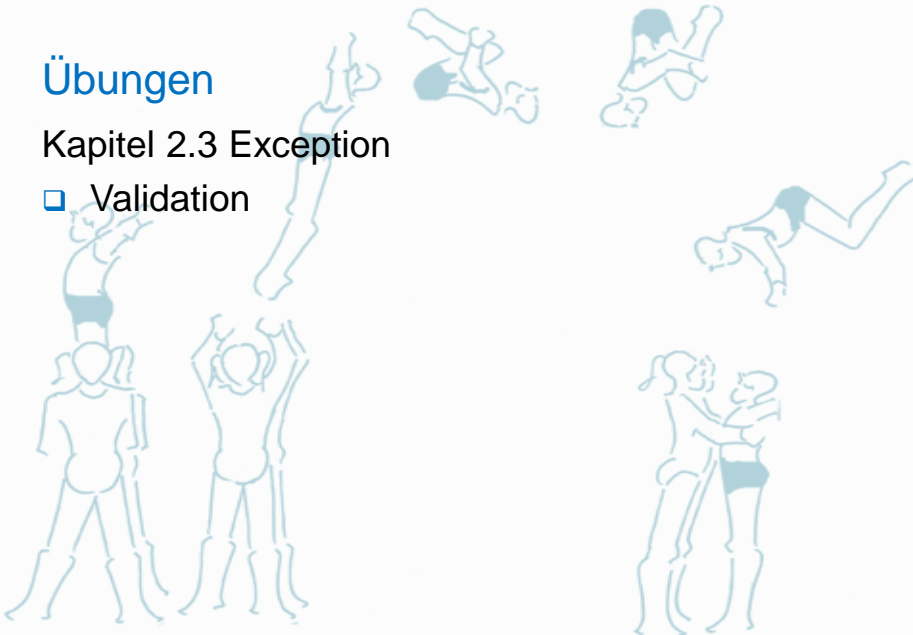
- ▶ Es stehen folgende **Built-in Exceptions** zur Verfügung
  - <https://docs.python.org/3/library/exceptions.html>
- ▶ Es können auch eigene Exceptions erstellt werden
  - Details siehe **User-defined Exceptions**  
<https://docs.python.org/3/tutorial/errors.html>

43

## Übungen

### Kapitel 2.3 Exception

#### ▣ Validation



44

## Abschnitt IV

## Funktionen &amp; Lambda

45

## Funktionen

```
def hello():  
    print("Hello")  
  
def greeting(name):  
    print("Hello", name)  
  
def weekend_greeting(name, greeting):  
    print("Hello %s, i wish you %s"%(name, greeting))  
  
hello()  
greeting("Tom")  
weekend_greeting("Zoé", "a nice weekend")
```

```
Hello  
Hello Tom  
Hello Zoé, i wish you a nice weekend
```

Output

46

## Funktionen mit Rückgabewert

```
def add(a, b):  
    return a + b  
  
def mean(values):  
    return sum(values) / len(values)  
  
res = add(3,9)  
print ("add(3,9) =", res)  
  
res = mean([4,8,12])  
print ("mean([4,8,12]) =", res)
```

```
add(3,9) = 12  
mean([4,8,12]) = 8.0
```

Output

47

## Positional und Keyword Argumente

```
def f1(arg1, arg2, arg3):  
    print("f1: arg1={}, arg2={}, arg3={}".format(arg1,arg2,arg3))  
  
def f2(arg1=None, arg2=10, arg3="Default"):  
    print("f2: arg1={}, arg2={}, arg3={}".format(arg1,arg2,arg3))  
  
def f3(arg1, arg2, arg3="Default", arg4=99):  
    print("f3: arg1={}, arg2={}, arg3={}, arg4={}".format(arg1,arg2,arg3,arg4))  
  
f1(1,2,3)  
f2()  
f2(arg2=22)  
f3(1,2)  
f3(1,2, arg4=88)
```

«Positional» Argumente sind obligatorisch und werden durch ihre Position zugeordnet

«Keyword» Argumente sind fakultativ und werden durch Position oder Name zugeordnet

«Mixed» Argumente werden durch die Position und Namen zugeordnet

```
f1: arg1=1, arg2=2, arg3=3  
f2: arg1=None, arg2=10, arg3=Default  
f2: arg1=None, arg2=22, arg3=Default  
f3: arg1=1, arg2=2, arg3=Default, arg4=99  
f3: arg1=1, arg2=2, arg3=Default, arg4=88
```

Output

- ▶ Weitere Möglichkeiten siehe Anhang
  - Argumente «entpacken» mit \*args und \*\*kwargs
  - Variable Anzahl Argumente mit \*args und \*\*kwargs

48



## Lambda Funktionen

- ▶ Ein Lambda Funktion ist eine kleine anonyme Funktion
  - mit einer beliebigen Anzahl Argumente
  - und einem Ausdruck (Expression)
- ▶ Syntax:  
**lambda** arguments : expression

```
add = lambda a, b : a + b  
  
print(add(1,2))
```

3

Output

## Lambda Funktionen II

- ▶ Lambda werden oft im Zusammenhang mit der Verarbeitung von Listen Elementen verwendet (z.B. mit map oder filter)
- ▶ Eine Lambda Funktion kann auch als Resultat einer anderen Funktion zurückgegeben werden
- ▶ Das ermöglicht elegante und mächtige Funktionen wie das folgende Beispiel zeigt:

```
def multipler(n):  
    return lambda a : a * n  
  
double = multipler(2)  
triple = multipler(3)  
  
print(double(10))  
print(triple(10))
```

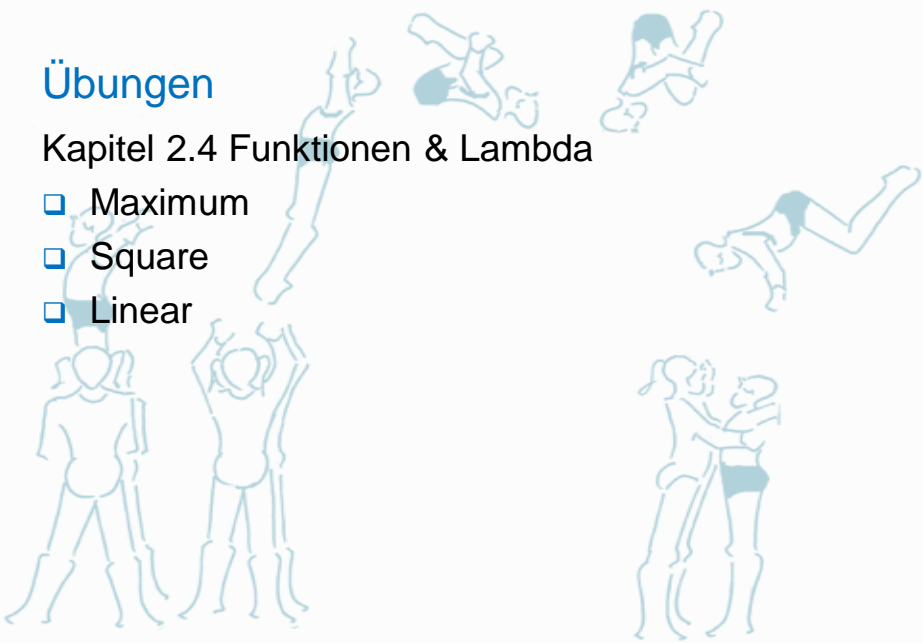
20  
30

Output

## Übungen

### Kapitel 2.4 Funktionen & Lambda

- ❑ Maximum
- ❑ Square
- ❑ Linear



The illustration shows several stylized human figures in various acrobatic poses. Some are standing with arms raised, while others are in mid-air or being supported by others, suggesting a group exercise or performance.

Copyright © iten-engineering.ch Python Einführung 51

51

## Kapitel 3

# Datenstrukturen



An empty rectangular box with a blue border, likely intended for a diagram or additional content related to data structures.

Copyright © iten-engineering.ch Python Einführung 52

52

## Abschnitt I

## Strings

53

## Deklaration

- ▶ Python 3 Strings sind in Unicode codiert
- ▶ Bei der Deklaration gibt es diverse Möglichkeiten:

```

quote      = "'Titanic' is a cool movie."
doublequote = '"Titanic' is a cool movie"
escape = '\m hungry'
multilines = """This is going
over multiples lines"""

path = "c:\\tmp"
rawpath = r"c:\tmp"          # Raw String - Escape Sequenzen werden ignoriert
unicode = '\U000000e4'       # ä (Python 3, wird direkt als Unicode interpretiert)
unicodePython2 = u'\U000000e4' # ä (Python 2)

```

54

## capitalize, format String

```
# capitalize
s = "this is a test"
print(s.capitalize())           # This is a test

# format
name = "Tom"
s = "Hello {}".format(name)     # Hello Tom
print(s)

s = "i have {} {}".format(1, "cat") # i have 1 cat
print(s)

s = "i have {1} {0}".format("cat", 1) # i have 1 cat
print(s)

s = "i have {count} {animal}".format(count=1, animal="cat") # i have 1 cat
print(s)
```

```
This is a test
Hello Tom
i have 1 cat
i have 1 cat
i have 1 cat
```

Output

Weitere Beispiele: <https://pyformat.info>

## format Nummer

```
s = "pi={:.2}".format(pi)           # {:breite.genauigkeit}
print("[", s, "]", sep="")

s = "pi={:10.2}".format(pi)
print("[", s, "]", sep="")

s = "pi={:>10.2}".format(pi)         # {ausrichtung:breite.genauigkeit}
print("[", s, "]", sep="")           # rechts

s = "pi={:<10.2}".format(pi)         # links
print("[", s, "]", sep="")

s = "pi={:^10.2}".format(pi)        # centriert
print("[", s, "]", sep="")

s = "pi={:0=10.2}".format(pi)       # {füllung=:breite.genauigkeit}
print("[", s, "]", sep="")
```

```
[pi=3.1]
[pi=      3.1]
[pi=      3.1]
[pi=3.1   ]
[pi=      3.1 ]
[pi=00000003.1]
```

Output

Weitere Beispiele: <https://pyformat.info>

## find, rfind, replace, starts/endswith, in

- ▶ `s.find(...)` gibt erste Position (Index) zurück
- ▶ `s.rfind(...)` gibt die letzte Position zurück

```
text = "this and that"
text.find("th")           # 0
text.rfind("th")          # 9

s = "Fall is coming".replace("Fall", "Winter") # Winter is coming

s.startswith("Winter")    # True
s.startswith("Fall")      # False

s.endswith("coming")      # True
s.endswith("leaving")     # False

"test" in "this is a test" # True
"x" in "abc"               # False
```

57

## split, join

```
text = "Hello Tom, how are you"

tokens = text.split()           # ["Hello", "Tom,", "how", "are", "you"]
print(tokens)

tokens = text.split(",")        # ["Hello Tom", "how are you"]
print(tokens)

num = 15.75
before, after = str(num).split(".")
print(before, ":", after)

tokens = ["Today", "we", "have", "a", "Python", "course."]
text = ".join(tokens)
print(text)
```

```
['Hello', 'Tom,', 'how', 'are', 'you']
['Hello Tom', ' how are you']
15 : 75
Today we have a Python course.
```

Output

58

## strip, lstrip,rstrip, ljust, center, rjust

```
# strip, lstrip, rstrip
" some text ".strip()      # "some text"
"_some text_".strip("_")   # "some text"
" some text ".lstrip()     # "some text "
" some text ".rstrip()     # " some text"

# ljust, center, rjust
s = "expand text".ljust(20)
print("[", s, "]", sep="")  # [expand text  ]

s = "expand text".rjust(20)
print("[", s, "]", sep="")  # [   expand text]

s = "expand text".center(20)
print("[", s, "]", sep="")  # [  expand text  ]
```

```
[expand text  ]
[   expand text]
[  expand text  ]
```

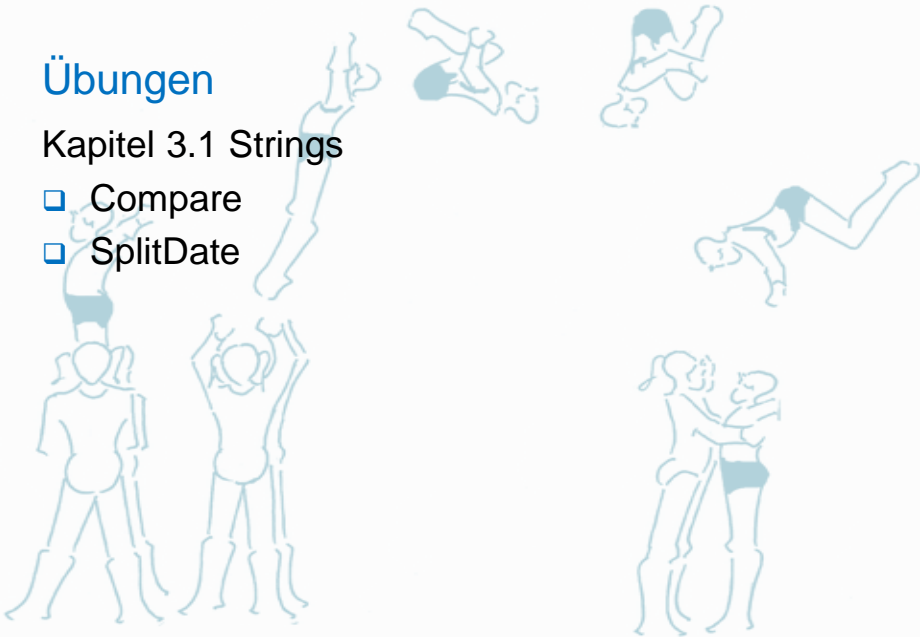
Output

59

## Übungen

### Kapitel 3.1 Strings

- ☐ Compare
- ☐ SplitDate



60

## Abschnitt II

## List &amp; Tuples

61

## List &amp; Tuples

- ▶ Tuple sind **immutable**, die Werte können nicht geändert werden, die Listengröße ist fix
- ▶ Listen sind **mutable**, Werte und die Anzahl Einträge einer Liste können dynamisch geändert werden
- ▶ Set und Dictionaries sind ebenfalls **mutable**

```
# Mutable List (Set, Dict):
l = [1,2,3]      # List class list() is mutable
l[1] = 22        # works fine

# Immutable Tuple:
t = (1,2,3)      # Tuple class tuple() is immutable
t[1] = 22        # raises TypeError
```

*Eselsbrücke für mutable: LSD (List, Set, Dictionaries)*

62

## Mutable vs. Immutable

- ▶ Veränderliche Objekte werden „in place“ verändert, welches alle Variablen beeinflusst die auf das Objekt zeigen
  - Um die Elemente einer Liste während einer Schleife zu modifizieren, sollte man auf das Element via Index zugreifen
  - Eine Liste über welche man iteriert sollte man nicht gleichzeitig modifizieren
- ▶ Für unveränderliche Objekte muss ein neues Objekt kreiert werden (jedes Mal wenn ein neuer Wert zu einer Variablen zugewiesen wird)
  - Unveränderliche Objekte werden während dem Iterieren nicht modifiziert
  - Wenn man ein unveränderliches Objekt über welches man iteriert modifiziert, kreiert es ein neues Objekt und hat keinen Einfluss auf die Schleife

63

## Initialisierung, len, min, max, count, index

```
# Initialisierung
l1 = []           # []
l2 = [1,2,3]      # [1, 2, 3]
l3 = list("abc")  # ['a', 'b', 'c']
l4 = list(range(4)) # [0,1,2,3]

# len, min, max, count, index
a = [9, 7, 9, 15, 12]
print(len(a))     # 5
print(min(a))     # 7
print(max(a))     # 15

print(a.count(9))  # 2      Gibt an wie oft (Anzahl) die 9 vorkommt
print(a.index(9))  # 0      Gibt den ersten Index der Zahl 9 an
print(a.index(12)) # 4      Gibt den Index der Zahl 12 an
```

Liste: ['X', 'Y', 'Z']  
 Index: 0 1 2

Letzter Index = len(Liste)-1

64



## all, any, in & not in

- ▶ `all(seq)`
  - Gibt True zurück, wenn **alle** Elemente der Sequenz `seq` True sind, sonst False
- ▶ `any(seq)`
  - Gibt True zurück, wenn **ein** oder **mehrere** Elemente der Sequenz `seq` True sind, sonst False
- ▶ `in` & `not in`
  - Prüfung ob eine Element in der Liste vorkommt oder nicht

```
numbers = [9, 7, 9, 15, 12]
elem = 7
if elem in numbers:
    print("list contains element:", elem)
elem = 99
if elem not in numbers:
    print("list does not contain element:", elem)
```

```
list contains element: 7
list does not contain element: 99
```

Output

## Elemente lesen und zuweisen

```
numbers = [1,2,3,4,5]
n = len(numbers)
print(type(numbers))
print(n)

print( numbers[0] ) # first element
print( numbers[-1] ) # last element
print( numbers[-2] ) # 2nd-last element
print( numbers[-n] ) # first element where n = len(numbers)

numbers[0] = 11 # change first element
print( numbers[0] )
```

```
<class 'tuple'>
5
1
5
4
1
11
```

Output

## Elemente hinzufügen

Befehl	Beschreibung
li.append(elem)	Fügt das Element elem am Ende der Liste li ein.
li.insert(pos, elem)	Fügt das Element elem an der Position pos der Liste li ein.
li.extend(sequence)	Fügt alle Elemente von sequence am Ende der Liste li ein.

```
data.append("Hello")
data += ["word"]
print(data)

data.insert(1, "wonderful")
print(data)

data.extend(["Life", "is", "awesome"])
print(data)
```

```
['Hello', 'word']
['Hello', 'wonderful', 'word']
['Hello', 'wonderful', 'word', 'Life', 'is', 'awesome']
```

Output

## Elemente entfernen

Befehl	Beschreibung
li.pop()	Das letzte Element entfernen und zurückgeben.
li.pop(pos)	Das Element an der Position pos entfernen und zurückgeben.
li.remove(elem)	Das Element elem entfernen.

```
items = [1, 2, 3, 4]
e = items.pop()
print(e)           # 4
print(items)       # [1, 2, 3]

e = items.pop(0)
print(e)           # 1
print(items)       # [2, 3]

items = [1, 2, 1, 7]
items.remove(1)
print(items)       # [2, 1, 7]
```

## Slicing

```
numbers = (1,2,3,4,5)
print( numbers[:3] ) # first 3 elements
print( numbers[-2:] ) # last 2 elements

print( numbers[::2] ) # every 2nd element
print( numbers[1:10:2] ) # every 2nd from 1 to 10
print( numbers[:] ) # all elements in sequence

first, second = numbers[:2]
print(first, second)
```

Slicing Syntax:  
numbers[von : bis : Schritt]

```
(1, 2, 3)
(4, 5)

(1, 3, 5)
(2, 4)
(1, 2, 3, 4, 5)

1 2
```

Output

## Slicing II

```
n = 20
r = list(range(n))
print("r      =", r)
print("r[1:-2:3] =", r[1:-2:3])

s = r[::3]
print("s      =", s)
print("s[1:-2] =", s[1:-2])

print("r[::3][1:-2]", r[::3][1:-2])
```

```
r      = [0, 1, 2, 3, 4, 5, 6, 7, 8, ..., 17, 18, 19]
r[1:-2:3] = [1, 4, 7, 10, 13, 16]

s      = [0, 3, 6, 9, 12, 15, 18]
s[1:-2] = [3, 6, 9, 12]

r[::3][1:-2] [3, 6, 9, 12]
```

Output

# sort

Befehl	Beschreibung
li.sort()	Die Liste li sortieren (reverse Parameter kontrolliert die Richtung)
sorted(li)	Gibt eine sortierte Liste zurück (reverse Parameter kontrolliert Richtung)

```
items = [1, 2, 0, 7]
items.sort()           # [0, 1, 2, 7] sortiert die Liste 'in place', gibt keine Objekt zurück
items.sort(reverse = True) # [7, 2, 1, 0]
sorted(items)          # [0, 1, 2, 7]
sorted(items,reverse = True) # [7, 2, 1, 0]

# Sort a String-List
words = "cheese, spam, egg, spam, bacon, spam"
sorted_words = sorted(words.split(", "))
sorted_words = ", ".join(sorted_words)
print(sorted_words)
```

bacon, cheese, egg, spam, spam, spam

Output

# map

- ▶ map(function, iterable)
  - Wendet die Funktion f auf jedes Element von iterable an.
  - In Python 3 wird ein Generator zurückgegeben

```
gen = map(abs, [-1, 2, -3])
for el in gen:
    print(el)
```

1
2
3

Output

```
list( map(abs, [-1, 2, -3]) )           # [1, 2, 3]
list( map(min, [(1,2),(8,5)]) )         # [1, 5]
list( map(sorted, [(1,5,3), (8,5,2)]) )  # [[1, 3, 5], [2, 5, 8]]
```

## filter

- ▶ `filter(function, iterable)`
  - behält alle Elemente `el` für welche `bool(function(el)) == True`.
  - In Python 3 wird ein Generator zurückgegeben

```
def isGreaterZero(x):  
    return x > 0;  
  
print( list( filter(isGreaterZero, [-1,0,1,2]) ) )
```

```
[1, 2]
```

Output

## Lambda

- ▶ Für jedes Filtering eine Funktion definieren macht wenig Sinn
- ▶ Die elegante Lösung ist eine **anonyme** lambda Funktion
- ▶ Kann direkt in einer **map** oder **filter** Anweisung definiert werden
- ▶ Die anonyme lambda wird eigentlich im namespace der Funktion einer Variablen zugewiesen
- ▶ Man kann eine lambda Funktion auch einer Variablen zuweisen

## Lambda II

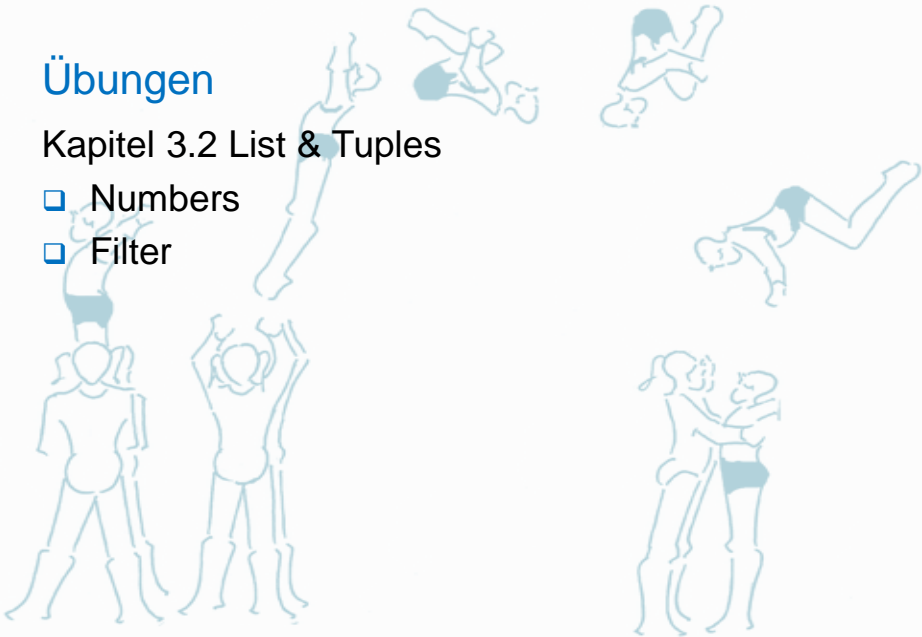
```
# Filtern mit lambda:  
data = [1, None, 2, 3]  
print(data)  
  
print (list(filter(lambda value: value is not None, data)))  
  
notNone = lambda value : value is not None  
print (list(filter(notNone, data)))
```

```
[1, None, 2, 3] Output  
  
[1, 2, 3]  
  
[1, 2, 3]
```

## Übungen

### Kapitel 3.2 List & Tuples

- ☐ Numbers
- ☐ Filter



## Abschnitt III

## Sets

77

## Sets

- Ein set ist eine ungeordnete Sammlung von eindeutigen Objekten.

```
# Initialisierung
s = set()           # {}
s = set([1,2,3])    # {1, 2, 3}
s = set([1,2,3,1])  # {1, 2, 3}
s = set("sam")       # {'s','a','m'}
```

```
# Set aus Liste erstellen
```

```
l1 = [1, 2, 3, 2, 6, 2]
s1 = set(l1)         # {1, 2, 3, 6}
```

Doppelte Einträge fallen weg.

78

## Add, Pop, Remove

```
# add
s = set([1,2,3])      # {1, 2, 3}
s.add(4)              # {1, 2, 3, 4}
s.add(0)              # {0, 1, 2, 3, 4}

# pop
e = s.pop()           # {1, 2, 3, 4}, e=0

# remove
s.remove(3)           # {1, 2, 4}
```

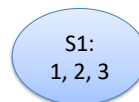
## Union, Intersection, Difference

```
s1 = set([1,2,3])
s2 = set([1,4])

# union
s = s1.union(s2)      # s = {1, 2, 3, 4}

# intersection
s = s1.intersection(s2) # s = {1}

# difference
s = s1.difference(s2)   # s = {2, 3}
s = s2.difference(s1)   # s = {4}
```





## Beispiel mit Schleife

```
l = [1, 1, 2, 3, 2]
print(l)

s1 = set()
for e in l:
    s1.add(e)
print(s1)      # {1, 2, 3}

s2 = {e for e in l}
print(s2)      # {1, 2, 3}
```

```
[1, 1, 2, 3, 2]
{1, 2, 3}
{1, 2, 3}
```

Output

## Abschnitt IV

### Dictionaries

## Dictionaries

- ▶ Ein Dictionary enthält **key:value** Paare und ist **nicht sortiert**
- ▶ Der **Zugriff** erfolgt via **key** (Schlüssel)

```
# Initialisierung mit Literal
d1 = {}
d2 = {"key": "value", 1: "value 1", 2: "value 2"}

# Initialisierung mit Klasse
d1 = dict()
d2 = dict([("key", "value of k"), (1, "value of 1"), (2, 4711)])
d3 = dict([("key", "value of k"), [1, "value of 1"], [2, 4711]])

# Zugriff via key
print(d2["key"])      # value of k
print(d2[1])          # value of 1
print(d2[2])          # 4711
```

Copyright © iten-engineering.ch

Python Einführung

83

83

## Dictionaries II

- ▶ Schlüssel können von **verschiedenen Typen** sein (ausser list und dict, da veränderlich)
  - int, float, complex, bool, str, tuple, function

```
d = {
    (1,2): "tuple value",
    print: "ok",
    "pi" : 3.14159,
    4711 : "Kölnisch Wasser"
}
print( d[(1,2)] )
print( d[print] )
print( d["pi"] )
print( d[4711] )
```

```
tuple value
ok
3.14159
Kölnisch Wasser
```

Output

Copyright © iten-engineering.ch

Python Einführung

84

84

## Insert, Update, Merge, Delete

```
# Neuer Wert zuweisen
d = {1: "H"}
d[1] = "Hello"
print(d)                                # {1: 'Hello'}

# Neues Element hinzufügen
d[2] = "World"
print(d)                                # {1: 'Hello', 2: 'World'}

# Update/Merge mit anderen Dictionary
dx = {2: "Tom", 3: "have a nice day"}
d.update(dx)
print(d)                                # Update existing entries, insert new entries
                                         # {1: 'Hello', 2: 'Tom', 3: 'have a nice day'}

# Wert löschen mit pop(key)
d.pop(3)
print(d)                                # {1: 'Hello', 2: 'Tom'}
```

## Keys, Values, Items

```
# Key, values, items:
d = {1:"v1", 2:"v2", 3:"v3"}
print(d.keys())                        # dict_keys([1, 2, 3])
print(d.values())                      # dict_values(['v1', 'v2', 'v3'])
print(d.items())                      # dict_items([(1, 'v1'), (2, 'v2'), (3, 'v3')])

# Iteration
d = {1:"v1", 2:"v2", 3:"v3"}

for k in d:
    print(k, "=", d[k])
```

```
dict_keys([1, 2, 3])
dict_values(['v1', 'v2', 'v3'])
dict_items([(1, 'v1'), (2, 'v2'), (3, 'v3')])

1 = v1
2 = v2
3 = v3
```

Output

# Schleifen

```
d = {1:"v1", 2:"v2", 3:"v3"}  
  
# Iterate over keys  
for k in d.keys():  
    print(k, "=", d[k])  
  
# Iterate over values  
for v in d.values():  
    print(v)  
  
# Iterate over items:  
for k, v in d.items():  
    print(k, "=", v)
```

```
1 = v1  
2 = v2  
3 = v3  
v1  
v2  
v3  
1 = v1  
2 = v2  
3 = v3
```

Output

87

# Übungen

- Kapitel 3.3 Sets
  - Mengen
- Kapitel 3.4 Dictionaries
  - l18N
  - Artikel

88

## Kapitel 4

# Klassen & Objekte

## Klassen & Objekte

- ▶ In Python ist alles ein Objekt: `"hello".upper()`
- ▶ Jede Klassenmethode benötigt die Instanzen Referenz `self` als ersten Parameter
  - Der Parameter Name kann frei gewählt werden
  - Als Konvention wird `self` als Name verwendet
- ▶ Der Typ der Attribute wird implizit bestimmt (wie bei allen anderen Variablen auch)
- ▶ Alle Konstruktoren haben den Namen `__init__`
- ▶ Der Destruktor wird automatisch durch Python aufgerufen (Garbage Collection)

## Klassen & Objekte II

```
class myClass:
    def __init__(self, arg1, arg2):
        self.var1 = arg1
        self.var2 = arg2

    def getVar1(self):
        return self.var1

# Instanz erstellen
myInstance = myClass("p1", "p2")

# Methode aufrufen (self is passed implicitly)
result = myInstance.getVar1()
```

91

## Konstruktor

- ▶ Typen von Konstruktoren
  - Default Konstruktor
  - Konstruktor mit Parametern
- ▶ Default Konstruktor
  - Einfacher Konstruktor ohne Argumente
  - Einzig die Instanz Referenz **self** muss definiert werden
- ▶ Konstruktor mit Parametern
  - Werden als «parameterized constructor» bezeichnet
  - Der **erster Parameter** ist die Instanz Referenz **self**
  - Die weiteren Parameter werden durch den Entwickler definiert

92

## Default Konstruktor

```
class GeekforGeeks:
```

```
    # default constructor
```

```
    def __init__(self):
        self.geek = "GeekforGeeks"
```

```
    # a method for printing data members
```

```
    def print_Geek(self):
        print(self.geek)
```

```
# creating object of the class
```

```
obj = GeekforGeeks()
```

```
# calling the instance method using the object obj
```

```
obj.print_Geek()
```

Instanz Referenz 'self' muss  
beim Aufruf nicht explizit  
angegeben werden!

```
GeekforGeeks
```

Output

93

## Konstruktor mit Parametern

```
class Addition:
```

```
    # parameterized constructor
```

```
    def __init__(self, f, s):
        self.first = f
        self.second = s
        self.answer = 0
```

```
    def display(self):
        print("First number = " + str(self.first))
        print("Second number = " + str(self.second))
        print("Addition of two numbers = " + str(self.answer))
```

```
    def calculate(self):
        self.answer = self.first + self.second
```

```
# creating object of the class, this will invoke parameterized constructor
```

```
obj = Addition(1000, 2000)
```

```
# perform Addition
```

```
obj.calculate()
```

```
# display result
```

```
obj.display()
```

```
First number = 1000
Second number = 2000
Addition of two numbers = 3000
```

Output

94

## Destruktor

- ▶ Destrukturen werden aufgerufen wenn eine Objekt Instanz zerstört wird.
- ▶ Sie sind das “Gegenstück” zum Konstruktor
- ▶ Der Destruktor wird automatisch durch Python aufgerufen, wenn die Garbage Collection die Instanz zerstört.
- ▶ Mit dem Befehl `del` kann dies auch explizit forciert werden.

## Destruktor II

```
class Item:

    # constructor
    def __init__(self, number):
        self.number = number
        print("Create item:", self.number)

    # destructor
    def __del__(self):
        print("Delete item:", self.number)

# create instance
item = Item(47)

# delete instance
del item
```

```
Create item: 47
Delete item: 47
```

Output



## Vererbung

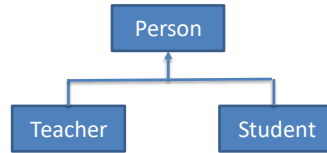
```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
        print(self.firstname, self.lastname)

class Teacher(Person):
    pass # Use pass keyword when you do not want to
        # add any other properties or methods to the class.

class Student(Person):
    def __init__(self, fname, lname, studentid):
        super().__init__(fname, lname)
        self.studentid = studentid

    def printname(self):
        print(self.firstname, self.lastname,
              self.studentid)
```



```
x = Person("John", "Doe")
x.printname()

x = Teacher("Mike", "Olsen")
x.printname()

x = Student("James", "Bond", "007")
x.printname()
```

```
John Doe
Mike Olsen
James Bond 007
```

Output

## Statische Attribute und Methoden

- ▶ Es ist auch möglich, statische Attribute und Methoden zu definieren
- ▶ Diese existieren 1x und gehören zur Klasse
- ▶ Der Aufruf erfolgt via Klassenname, d.h. man benötigt keine Instanz für den Zugriff.
- ▶ Die Klassenattribute werden auf **Klassen Level** definiert.
- ▶ Die Methoden werden mit **@staticmethod** gekennzeichnet

Mit @classmethod gibt es in Python noch eine weitere Abstufung.

Details siehe: <https://rapid.wordpress.com/2008/07/02/python-staticmethod-vs-classmethod>

## Statische Attribute und Methoden II

```
class Counter:
    instance_count = 0

    def __init__(self):
        Counter.instance_count += 1

    @staticmethod
    def print(action):
        print(action, "instance. Count:", Counter.instance_count)

    def __del__(self):
        type(self).instance_count -= 1

if __name__ == "__main__":
    counters = []
    for i in range(4):
        counter = Counter()
        counters.append(counter)
        Counter.print("Created")
    while len(counters) > 0:
        counter = counters.pop()
        del counter
        Counter.print("Deleted")
```

Statisches Attribut und  
statische Methode

Output

```
Created instance. Count: 1
Created instance. Count: 2
Created instance. Count: 3
Created instance. Count: 4
Deleted instance. Count: 3
Deleted instance. Count: 2
Deleted instance. Count: 1
Deleted instance. Count: 0
```

## Übungen

### Kapitel 4. Klassen & Objekte

- ☐ Kreis
- ☐ Zylinder
- ☐ Fahrzeug

### Optionales Thema Unit Testing

- ☐ MathUtil
- ☐ UnitTest

# Kapitel 5

## File Input/Output

## File Input/Output

Befehl	Beschreibung
<b>Öffnen und Schliessen</b>	
f = open("path/to/file", "r")	Datei öffnen zum Lesen
f = open("path/to/file", "w")	Datei öffnen zum Schreiben
f = open("path/to/file", "a")	Datei öffnen zum Anhängen
f.close()	Datei schliessen
<b>Lesen</b>	
s = f.read()	Ganzer Dateiinhalt lesen und Rückgabe String
s = f.readline()	Nächste Zeile der Datei lesen und Rückgabe String
l = f.readlines()	Alle Zeilen der Datei lesen und Rückgabe Liste
<b>Schreiben</b>	
f.write("Hello World")	String in Datei schreiben
f.writelines(["Hello", "World"])	Liste in Datei schreiben

## Auto Close

- Datei öffnen und am Ende automatisch schliessen:

```
with open("test-numbers.txt", "w") as f:
    f.writelines(["1.0\n", "1.5\n", "2.0\n"])

with open("hello.txt", "r") as f:
    text = f.read()

print(text)
```

Hello World.

Output

103

## CSV Files

```
import csv
titles = ["firstname", "lastname"]
rows = [['Pipi', 'Langstrumpf'], ['Peter', 'Pan'], ['Marie', 'Fischer']]

with open("test.csv", "w") as f:
    writer = csv.writer(f)
    writer.writerow(titles)
    writer.writerows(rows)

data = []
with open("test.csv", "r") as f:
    reader = csv.reader(f)
    titles = reader.__next__()
    for row in reader:
        if (len(row) > 0):
            data.append(row)
    print(data)
```

Das csv Modul funktioniert mit reader/writer Objekten

[['Pipi', 'Langstrumpf'], ['Peter', 'Pan'], ['Marie', 'Fischer']]

Output

104

## CSV Files mit Pandas

```
import pandas

# read a csv file
with open("test.csv", "r") as f:
    data_frame = pandas.read_csv(f)

# write the data_frame to a csv file
data_frame.index.name = "index"
with open("test-pandas.csv", "w") as f:
    data_frame.to_csv(f)
```

```
index,firstname,lastname                                test-pandas.csv
0,Pipi,Langstrumpf
1,Peter,Pan
2,Marie,Fischer
```

- Im Kapitel «Data Science Libraries» folgen weitere Informationen zum Pandas Framework

## Read/Write Objects

- Pickle kann Python Objekte lesen und speichern:

```
import pickle

d = {1:2, "k":[1,2,3], "fun":print}
print(d)

with open("test-dict.pkl", "wb") as fout:    # open file for write-binary
    pickle.dump(d, fout)                    # dump pickle file

with open("test-dict.pkl", "rb") as fin:    # open file for read-binary
    d2 = pickle.load(fin)                  # load pickle file
print(d2)
```

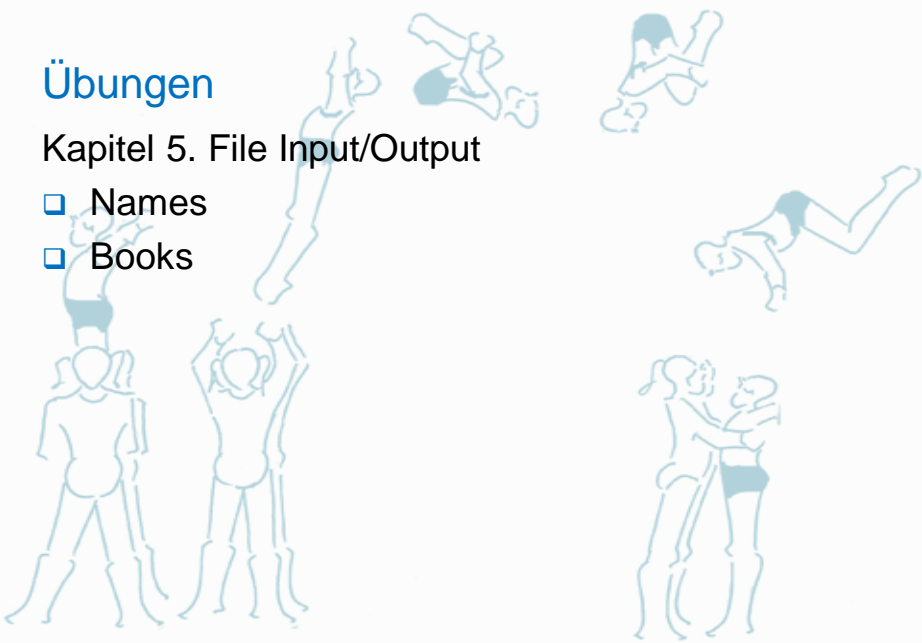
```
{1: 2, 'k': [1, 2, 3], 'fun': <built-in function print>}
{1: 2, 'k': [1, 2, 3], 'fun': <built-in function print>}
```

Output

# Übungen

## Kapitel 5. File Input/Output

- ▣ Names
- ▣ Books



Copyright © iten-engineering.ch Python Einführung 107

107

## Kapitel 6

# Module & Packages

Copyright © iten-engineering.ch Python Einführung 108

108

## Abschnitt I

### Module

109

### Module

- ▶ Mit Python können Definitionen (Funktionen, Klassen) in eigene Dateien (Module) ausgelagert werden.
- ▶ Die Definitionen eines Moduls können in andere Module oder das Hauptprogramm importiert und dort genutzt werden.
- ▶ Der Datei Name entspricht dabei dem Modulnamen mit dem Suffix «.py».
- ▶ Innerhalb vom Modul ist der Modulname via die interne Variable "\_\_name\_\_" verfügbar.

110

## Modul fibo.py

```
def print_fib(n):
    """ write Fibonacci series up to n """
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib(n):
    """ return Fibonacci series up to n """
    a, b = 0, 1
    result = []
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

fibo.py

111

## Verwendung vom Modul fibo.py

```
import fibo

print ("Fibo sample:")
fibo.print_fib(100)

result = fibo.fib(100)
print(result)

print(("Show module details:"))
print(dir(fibo))
```

```
Fibo sample:
0 1 1 2 3 5 8 13 21 34 55 89
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

Show module details:
['_builtins_', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', 'fib',
'print_fib']
```

Output

112



## Import

- ▶ Sample: `import module`
  - imports everything and keeps it in the module's namespace
  - `module.func()`
  - `module.className.func()`
- ▶ Sample: `from module import *`
  - imports everything under the current namespace
  - `func()`
  - `className.func()`
  - **not recommended**
- ▶ Sample: `from module import className`
  - selectively imports under the current namespace
  - `className.func()`
  - like standard modules: `math`, `os`, `sys`

113

## Import with custom name

```

if visual_mode:
    # in visual mode, we draw using graphics
    import draw_visual as draw
else:
    # in textual mode, we print out text
    import draw_textual as draw

def main():
    result = play_game()
    # this can either be visual or textual depending on visual_mode
    draw.draw_game(result)

...

```

game.py

114

## Abschnitt II

### Packages

115

### Packages

- ▶ Mit Hilfe von Packages werden Python Module organisiert.
- ▶ Dabei kommt eine Punkt Notation zum Einsatz.
- ▶ Mit dem Ausdruck A.B wird zum Beispiel das Modul B innerhalb vom Package A referenziert.
- ▶ Jedes Modul hat seinen eigenen Namensraum, dadurch werden Namenskonflikte (von Variablen und Klassen) verhindert

116

## Packages II

- ▶ Packages sind Verzeichnisse, die **Module** der Packages sind in entsprechenden **Unterverzeichnissen** organisiert.
- ▶ Innerhalb vom **Package** Verzeichnis muss zwingend eine Datei **`__init__.py`** vorhanden sein, damit Python das Verzeichnis als Package identifiziert
- ▶ Wenn ein Package **importiert** wird, durchsucht Python die Verzeichnisse der **`sys.path`** Variable, um das entsprechende Package Verzeichnis zu lokalisieren

117

## Beispiel Package Struktur

<code>sound/</code>	<i>Top-level package</i>
<code>__init__.py</code>	<i>Initialize the sound package</i>
<code>formats/</code>	<i>Subpackage for file format conversions</i>
<code>__init__.py</code>	
<code>wavread.py</code>	<i>&gt; Module wavread</i>
<code>wavwrite.py</code>	<i>&gt; Module wavwrite</i>
<code>...</code>	
<code>effects/</code>	<i>Subpackage for sound effects</i>
<code>__init__.py</code>	
<code>echo.py</code>	<i>&gt; Module echo</i>
<code>surround.py</code>	
<code>reverse.py</code>	
<code>...</code>	
<code>filters/</code>	<i>Subpackage for filters</i>
<code>__init__.py</code>	
<code>equalizer.py</code>	
<code>vocoder.py</code>	
<code>karaoke.py</code>	

118

## Import

- ▶ Import Modul und Referenzierung mit vollen Namen

```
import sound.effects.echo
sound.effects.echo.echofilter(input, output, delay=0.7, ...)
```

- ▶ Import Modul ohne Package Prefix und Referenzieren mit einfachem Modulnamen

```
from sound.effects import echo
echo.echofilter(input, output, delay=0.7, atten=4)
```

- ▶ Direkter Import einer Funktion oder Variablen eines Moduls

```
from sound.effects.echo import echofilter
echofilter(input, output, delay=0.7, atten=4)
```

119

## Import mit \*

- ▶ Bei einem Import mit \* werden die Module importiert, welche im Package entsprechend angegeben werden
  - Die Angabe erfolgt in der Datei `__init__.py` mit Hilfe der Variable `__all__`
- ▶ Diese definiert eine Liste der Module, die das Package zur Verfügung stellt, wie zum Beispiel:
  - `__all__ = ["echo", "surround", "reverse"]`
  - Mit dem folgenden Import Statement würden somit die obigen Module auf einmal importiert:  
`sound.effects import *`

120

## Import mit \* falls \_\_all\_\_ nicht definiert ist

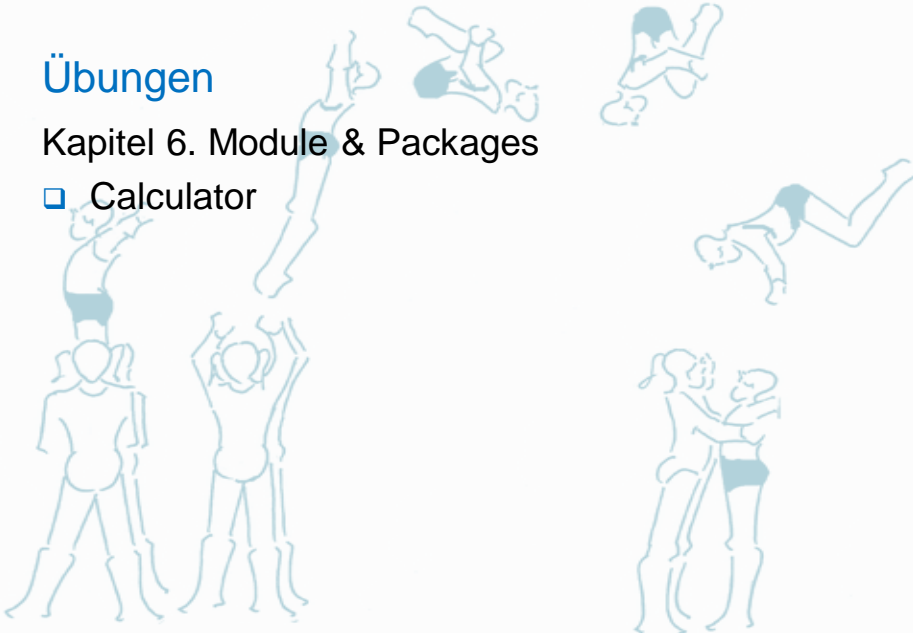
- ▶ Es ist in der Verantwortung der Package Entwickler bei neuen Package Release, die Liste **up-to-date** zu halten
  - Es ist auch möglich keine Angaben zu machen (falls man keine Verwendung für den Import mit \* hat)
- ▶ Falls **\_\_all\_\_** nicht definiert ist, werden mit dem Statement **sound.effects import \*** die **Module** von **sound.effects** nicht importiert
  - Es wird nur sichergestellt dass das Package sound.effects importiert wird
  - Der Initialisierungs Code der Datei **\_\_init\_\_.py** ausgeführt wird sowie die dort definierten Namen importiert werden
- ▶ Generell werden Imports mit **expliziter** Angabe der Module **empfohlen**

121

## Übungen

### Kapitel 6. Module & Packages

#### ❑ Calculator



122

## Kapitel 7

### Standard Libraries

123

## Abschnitt I

### math

124

## math

Methode	Beschreibung
<a href="#">math.acos()</a>	Returns the arc cosine of a number
<a href="#">math.acosh()</a>	Returns the inverse hyperbolic cosine of a number
<a href="#">math.asin()</a>	Returns the arc sine of a number
<a href="#">math.asinh()</a>	Returns the inverse hyperbolic sine of a number
<a href="#">math.atan()</a>	Returns the arc tangent of a number in radians
<a href="#">math.atan2()</a>	Returns the arc tangent of y/x in radians
<a href="#">math.atanh()</a>	Returns the inverse hyperbolic tangent of a number
<a href="#">math.ceil()</a>	Rounds a number up to the nearest integer
<a href="#">math.comb()</a>	Returns the number of ways to choose k items from n items without repetition and order
<a href="#">math.copysign()</a>	Returns a float consisting of the value of the first parameter and the sign of the second parameter

Quelle: [https://www.w3schools.com/python/module\\_math.asp](https://www.w3schools.com/python/module_math.asp)

## math II

Methode	Beschreibung
<a href="#">math.cos()</a>	Returns the cosine of a number
<a href="#">math.cosh()</a>	Returns the hyperbolic cosine of a number
<a href="#">math.degrees()</a>	Converts an angle from radians to degrees
<a href="#">math.dist()</a>	Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
<a href="#">math.erf()</a>	Returns the error function of a number
<a href="#">math.erfc()</a>	Returns the complementary error function of a number
<a href="#">math.exp()</a>	Returns E raised to the power of x
<a href="#">math.expm1()</a>	Returns $E^x - 1$
<a href="#">math.fabs()</a>	Returns the absolute value of a number
<a href="#">math.factorial()</a>	Returns the factorial of a number
<a href="#">math.floor()</a>	Rounds a number down to the nearest integer

### math III

Methode	Beschreibung
<a href="#">math.log()</a>	Returns the natural logarithm of a number, or the logarithm of number to base
<a href="#">math.log10()</a>	Returns the base-10 logarithm of x
<a href="#">math.log1p()</a>	Returns the natural logarithm of 1+x
<a href="#">math.log2()</a>	Returns the base-2 logarithm of x
<a href="#">math.perm()</a>	Returns the number of ways to choose k items from n items with order and without repetition
<a href="#">math.pow()</a>	Returns the value of x to the power of y
<a href="#">math.prod()</a>	Returns the product of all the elements in an iterable
<a href="#">math.radians()</a>	Converts a degree value into radians
<a href="#">math.remainder()</a>	Returns the closest value that can make numerator completely divisible by the denominator
<a href="#">math.sin()</a>	Returns the sine of a number

### math VI

Methode	Beschreibung
<a href="#">math.sin()</a>	Returns the sine of a number
<a href="#">math.sinh()</a>	Returns the hyperbolic sine of a number
<a href="#">math.sqrt()</a>	Returns the square root of a number
<a href="#">math.tan()</a>	Returns the tangent of a number
<a href="#">math.tanh()</a>	Returns the hyperbolic tangent of a number
<a href="#">math.trunc()</a>	Returns the truncated integer parts of a number
Konstanten	Beschreibung
<a href="#">math.e</a>	Returns Euler's number (2.7182...)
<a href="#">math.inf</a>	Returns a floating-point positive infinity
<a href="#">math.nan</a>	Returns a floating-point NaN (Not a Number) value
<a href="#">math.pi</a>	Returns PI (3.1415...)
<a href="#">math.tau</a>	Returns tau (6.2831...)



## cmath

- ▶ Neben den gezeigten Math Funktionen bietet Python mit dem `cmath` Modul auf Funktionen für **komplexe Zahlen**.
- ▶ Die `cmath` Methoden unterstützen
  - `int`, `float` und `complex` Zahlen sowie
  - Objekte mit `__complex__()` oder `__float__()` Methoden

129

## Abschnitt II

OS

130

## OS

- ▶ Mit dem **os** Modul werden diverse “Operation System” Methoden unterstützt.
- ▶ Dazu gehören zum Beispiel Methoden zum Auflisten oder Erstellen von Dateien und Verzeichnissen
- ▶ Oder zum Ausführen von Shell Kommandos

```
# List files of current directory on linux/unix  
os.system('ls -al')
```

131

## os Beispiele

```
# Changing the Current Working Directory  
os.chdir(demo_dir)
```

```
print(os.getcwd())
```

```
# Create some files
```

```
open('fileA.txt', 'a').close()
```

```
open('fileB.txt', 'a').close()
```

```
open('fileC.txt', 'a').close()
```

```
# List Files and Sub-directories
```

```
print(os.listdir())
```

```
..\example\07-std-libs\os-demo  
['fileA.txt', 'fileB.txt', 'fileC.txt']
```

Output

132

## Abschnitt III

### sys, subprocess

133

### sys

- ▶ Mit sys können diverse System Befehle ausgeführt werden oder das Programm mit exit() verlassen werden
- ▶ Im weiteren können die geladenen Python Module abgefragt werden oder die Version des Python Interpreters abgefragt werden
- ▶ Die folgende Tabelle zeigt die wichtigsten Befehle in der Kurzübersicht

134

sys

Methode	Beschreibung
argv	Enthält die Script/Programm Argumente. <ul style="list-style-type: none"><li>• Index 0 beinhaltet den Script Namen</li><li>• Anschliessend folgen die Argumente</li></ul>
executable	Ausgabe absoluter Pfad zum Python Interpreter.
exit	Mit exit kann das Programm, zum Beispiel nach einer Exception, kontrolliert verlassen werden. <ul style="list-style-type: none"><li>• Dabei kann ein Code (in der Regel ein Integer) mitgegeben werden.</li><li>• Der Code 0 steht für eine erfolgreiche Beendigung.</li></ul>
modules	Dictionary mit allen Python Modulen die der Interpreter seit dem Start geladen hat
path	Liste mit Verzeichnispfaden in der nach Modulen gesucht wird. <ul style="list-style-type: none"><li>• Die Initialisierung erfolgt Anhand der Installation sowie der Umgebungsvariablen PYTHONPATH.</li><li>• Mit append können weitere Verzeichnisse angehängt werden</li></ul>

Quelle: [https://python101.pythonlibrary.org/chapter20\\_sys.html](https://python101.pythonlibrary.org/chapter20_sys.html)

sys

Methode	Beschreibung
platform	Ausgabe der Plattform Identifier ('win32', etc.). Kann zum Beispiel für Plattform spezifische Imports verwendet werden.
stdin / stdout / stderr	Standard Input-, Output- und Errorstreams des Interpreters.
version	Ausgabe der Version des Python Interpreters

```
os = sys.platform
if os == "win32":
    # use Window-related code here
    import _winreg
elif os.startswith('linux'):
    # do something Linux specific
    import subprocess
    subprocess.Popen(["ls, -l"])
```

Beispiel mit sys.platform

## subprocess

- ▶ Mit subprocess können Prozesse oder andere Programme von Python aus gestartet werden
- ▶ Das Modul wurde mit Python 2.4 eingeführt als Ersatz für os.popen, os.spawn und os.system

```
import subprocess
```

```
code = subprocess.call("notepad.exe")
```

```
if code == 0:  
    print("Success!")  
else:  
    print("Error!")
```

Beispiel

## Abschnitt IV

### re (Regular Expression)

## Regular Expression

- ▶ Ein regulärer Ausdruck ist eine Sequenz von Zeichen die ein Muster (Pattern) definieren
- ▶ Damit kann Strings nach Muster durchsuchen oder prüfen ob diese ein bestimmtes Format erfüllen, wie zum Beispiel:
  - Kreditkarten Format
  - AHV Nummer
  - ISBN Nummer
  - E-Mail Adresse
- ▶ In Python wird dazu das Modul `re` verwendet

## Methoden

- ▶ Das `re` Modul verfügt über folgende Methoden:

Methode	Beschreibung
<a href="#"><code>findall</code></a>	Returns a list containing all matches
<a href="#"><code>search</code></a>	Returns a <a href="#"><code>Match object</code></a> if there is a match anywhere in the string
<a href="#"><code>split</code></a>	Returns a list where the string has been split at each match
<a href="#"><code>sub</code></a>	Replaces one or many matches with a string

- ▶ The `Match` object has properties and methods used to retrieve information about the search, and the result:
  - `span()` returns a tuple containing the start-, and end positions of the match.
  - `string` returns the string passed into the function
  - `group()` returns the part of the string where there was a match

## findall()

*# Print a list of all matches:*

```
txt = "The rain in Spain"  
x = re.findall("ai", txt)
```

```
print(x)
```

*# Return an empty list if no match was found:*

```
txt = "The rain in Spain"  
x = re.findall("Portugal", txt)
```

```
print(x)
```

```
['ai', 'ai']  
[]
```

Output

## search()

*# Search for the first white-space character in the string:*

```
txt = "The rain in Spain"  
x = re.search("\s", txt)
```

```
print("The first white-space character is located in position:", x.start())
```

*# Make a search that returns no match:*

```
txt = "The rain in Spain"  
x = re.search("Portugal", txt)
```

```
print(x)
```

```
The first white-space character is located in position: 3  
None
```

Output

## split()

*# Split at each white-space character:*

```
txt = "The rain in Spain"
```

```
x = re.split("\s", txt)
```

```
print(x)
```

*# Split the string only at the first occurrence:*

```
txt = "The rain in Spain"
```

```
x = re.split("\s", txt, 1)
```

```
print(x)
```

```
['The', 'rain', 'in', 'Spain']  
['The', 'rain in Spain']
```

Output

## sub()

*# Replace every white-space character with the number 9:*

```
txt = "The rain in Spain"
```

```
x = re.sub("\s", "9", txt)
```

```
print(x)
```

*# Replace the first 2 occurrences:*

```
txt = "The rain in Spain"
```

```
x = re.sub("\s", "9", txt, 2)
```

```
print(x)
```

```
The9rain9in9Spain  
The9rain9in Spain
```

Output



## Meta Character

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
\$	Ends with	"world\$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
	Either or	"falls   stays"
()	Capture and group	

## Special Sequences

Character	Description	Example
\A	Returns a match if the specified characters are at the <b>beginning</b> of the string	"\AThe"
\b	Returns a match where the specified characters are at the <b>beginning or at the end of a word</b> (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are <b>present, but NOT at the beginning (or at the end) of a word</b> (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string <b>contains digits</b> (numbers from 0-9)	"\d"
\D	Returns a match where the string <b>DOES NOT contain digits</b>	"\D"

## Special Sequences II

Character	Description	Example
\s	Returns a match where the string <b>contains a white space</b> character	"\s"
\S	Returns a match where the string <b>DOES NOT contain a white space</b> character	"\S"
\w	Returns a match where the string contains <b>any word</b> characters (characters from a to Z, digits from 0-9, <b>and the underscore _</b> character)	"\w"
\W	Returns a match where the string <b>DOES NOT contain any word characters</b>	"\W"
\Z	Returns a match if the specified characters are at <b>the end of the string</b>	"Spain\Z"


## Sets

Set	Description
[arn]	Returns a match where <b>one of the specified characters</b> (a, r, or n) are <b>present</b>
[a-n]	Returns a match for any lower case <b>character</b> , alphabetically between a and n
[^arn]	Returns a match for any character <b>EXCEPT a, r, and n</b>
[0123]	Returns a match where any of the <b>specified digits</b> (0, 1, 2, or 3) are <b>present</b>
[0-9]	Returns a match for any <b>digit between 0 and 9</b>
[0-5][0-9]	Returns a match for any <b>two-digit numbers from 00 and 59</b>
[a-zA-Z]	Returns a match for any character alphabetically between a and z, <b>lower case OR upper case</b>
[+]	In sets, +, *, .,  , (, ), \$, {} has no special meaning, so [+] means: return a match for any + character in the string

# Übungen

## Kapitel 7. Standard Libraries

- ▣ Directory
- ▣ Directories
- ▣ RegEx



Copyright © iten-engineering.ch Python Einführung 149

149

## Kapitel 8

# Data Science Libraries

Copyright © iten-engineering.ch Python Einführung 150

150

Abschnitt I

Übersicht

151

Data Science Libraries

- ▶ Python ist (neben R) für viele die erste Wahl für Data Science Aufgaben.
- ▶ Es gibt eine Vielzahl an Bibliotheken für die verschiedenen Statistik und Data Science Bereiche

Bereich	Bibliothek
Data Mining	Scrapy, Bautiful Soap
Data Processing and Modeling	NumPy, SciPy, Pandas, Keras, SciKit Learn, PyTorch, TensorFlow, XGBoost
Data Visualization	Mathplotlib, Seaborn, Bokeh, Plotly, pydot

152

## NumPy

- ▶ NumPy steht für **Numerical Python** und ist ein perfektes Werkzeug für wissenschaftliches Arbeiten
- ▶ Die Bibliothek bietet viele praktische Features für **Operationen mit Arrays** die Werte vom desselben Datentypen speichern
- ▶ Mathematische Operationen mit **NumPy Arrays** (sowie deren Vektorisierung) sind **effizient** und **Leistungstark**
- ▶ Viele **weitere Bibliotheken basieren** auf NumPy

## SciPy

- ▶ **SciPy** ist eine Sammlung von mathematischen Funktionen und Algorithmen und **basiert auf NumPy**
- ▶ Es bietet **effiziente Routinen** für Optimierungen, Integration und andere Aufgaben
- ▶ SciPy ist in Sub **Packages organisiert**, welche normalerweise individuell importiert und verwendet werden

```
import numpy as np
from scipy import optimize

x = np.arange(0,10)
y = 2 * x + 3 + np.random.random(10)
res = optimize.curve_fit(lambda x, a, b: a*x + b, x, y)
```

## SciPy Packages

- ▶ Special functions (scipy.special)
- ▶ Integration (scipy.integrate)
- ▶ Optimization (scipy.optimize)
- ▶ Interpolation (scipy.interpolate)
- ▶ Fourier Transforms (scipy.fft)
- ▶ Signal Processing (scipy.signal)
- ▶ Linear Algebra (scipy.linalg)
- ▶ Statistics (scipy.stats)
- ▶ Image processing (scipy.ndimage)
- ▶ File IO (scipy.io)
- ▶ Sparse eigenvalue with ARPACK (scipy.sparse)
- ▶ Compressed Sparse Graph (scipy.sparse.csgraph)
- ▶ Spatial data and algorithms (scipy.spatial)

Reference: <https://docs.scipy.org/doc/scipy/reference>

155

## Pandas

→ Details  
Abschnitt III

- ▶ **Pandas** ist das Python Modul für Datenanalyse und Manipulation und **basiert** ebenfalls **auf NumPy**
  - Datensätze vereinen (merge)
  - Daten gruppieren (d.h. in Gruppen aufsplitten)
  - Operationen auf Gruppen ausführen
  - Label-based indexing und slicing
- ▶ Stellt zwei Klassen zur Verfügung:
  - 1-Dimensionale Daten (Series)
  - 2-Dimensionale Daten mit Labels (DataFrame)

156

## SciKit Learn

- ▶ Der Name SciKit entstand aus den Wörtern **SciPy** und **Toolkit**
- ▶ SciKit Learn basiert auf SciPy und ergänzt diese mit Machine Learning Funktionen für:
  - Supervised-learning (Classification, Regression)
  - Unsupervised-learning (Clustering, Density-estimation)
  - Data Preparation (Preprocessing, Feature extraction, Feature selection)
  - Estimator scoring

## Keras

- ▶ Keras ist eine Open Source Bibliothek im Bereich **Deep-Learning**
- ▶ Es können **neuronaler Netzwerke** modelliert und trainiert werden
- ▶ Keras bietet eine **einheitliche Schnittstelle** zu weiteren Frameworks wie TensorFlow, Microsoft Cognitive Toolkit oder Theano
- ▶ In zukünftigen Releases soll Keras in erster Linie auf TensorFlow ausgerichtet werden

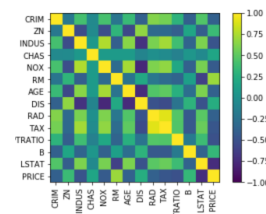
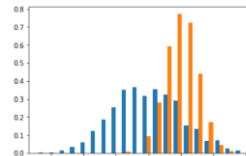
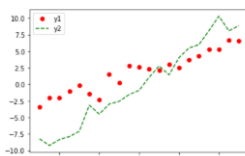
## TensorFlow

- ▶ TensorFlow ist ein Python Framework für **Machine Learning** und **Deep Learning**
- ▶ Es ist das beste Werkzeug für die Themenfelder der **Objektidentifikation** und **Spracherkennung**
- ▶ Es hilft bei der Arbeit mit künstlichen **neuronalen Netzwerken**, die mehrere Datensätze verarbeiten müssen.
- ▶ TensorFlow wird ständig erweitert, so u.a. bei
  - Korrekturen von potenziellen Sicherheitslücken
  - Verbesserungen bei der Integration von GPU Support

## Mathplotlib

→ Details  
siehe Übungen

- ▶ Die **matplotlib** Bibliothek ist das Python Modul zur grafischen Darstellung von Daten.
- ▶ Es ist ähnlich zu den grafischen Funktionen von Matlab
- ▶ Das Modul **matplotlib.pyplot** enthält die Funktionen um Grafiken zu erstellen





## Abschnitt II

## NumPy

161

## NumPy

- ▶ Die **NumPy** Bibliothek stellt die Basisobjekte für technisches Rechnen in Python zur Verfügung
  - Array
  - Matrix
- ▶ Es hat u.a. Numpy Funktionen für **Mathematik**, **Statistik** und **lineare Algebra**
- ▶ Der Import erfolgt i.d.R. mit dem Kürzel **np**
- ▶ Numpy wird oft als **Basis** für andere Bibliotheken genutzt (z.B. bei Pandas)

```
import numpy as np
```

```
b = np.array([6, 7, 8])
```

```
print(b) # array([6, 7, 8])
```

```
print(type(b)) # <class 'numpy.ndarray'>
```

162

## Array information

Methode	Beschreibung
ndim	The number of axes (dimensions) of the array
shape	Tuple of integers indicating the size of the array in each dimension. <ul style="list-style-type: none"><li>for a matrix with n rows and m columns, shape will be (n,m).</li><li>the length of the shape tuple is therefore the number of axes</li></ul>
size	Total number of elements of the array.
dtype	Object describing the type of the elements in the array. <ul style="list-style-type: none"><li>The type can be specified (Standard Python or Numpy types).</li></ul>
itemsize	The size in bytes of each element of the array. <ul style="list-style-type: none"><li>For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8).</li><li>It is equivalent to ndarray.dtype.itemsize.</li></ul>
data	The buffer containing the actual elements of the array. <ul style="list-style-type: none"><li>Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.</li></ul>
reshape	reshape array elements to the given shape

163

## Array information II

```
a = np.arange(15).reshape(3,5)
print(a)           # [[ 0,  1,  2,  3,  4],
                   # [ 5,  6,  7,  8,  9],
                   # [10, 11, 12, 13, 14]]
print(a.shape)     # (3, 5)
print(a.ndim)      # 2
print(a.dtype.name) # 'int64'
print(a.itemsize)  # 8
print(a.size)      # 15
print(type(a))     # <class 'numpy.ndarray'>
```

```
[ 0  1  2  3  4]
[ 5  6  7  8  9]
[10 11 12 13 14]]
(3, 5)
2
int32
4
15
<class 'numpy.ndarray'>
```

Output

164

## Array creation

```
a = np.array(1,2,3,4) # WRONG, Traceback (most recent call last):
                        # TypeError: array() takes from 1 to 2 positional arguments
                        # but 4 were given

a = np.array([1,2,3,4]) # RIGHT
print(a)

b = np.array([(1.5,2,3), (4,5,6)]) # [[1.5, 2., 3.],
print(b)                          # [4., 5., 6.]]

# The type of the array can also be explicitly specified at creation time:
c = np.array([ [1,2], [3,4] ], dtype=complex )

print(c)                      # [[1.+0.j, 2.+0.j],
                              # [3.+0.j, 4.+0.j]]
```

165

## Array creation II

- ▶ Often, the elements of an array are originally unknown, but its size is known.
- ▶ Hence, NumPy offers several functions to **create arrays** with initial placeholder content.
- ▶ These minimize the necessity of growing arrays, an expensive operation.
  - **zeros**: the function zeros creates an array full of zeros,
  - **ones**: the function ones creates an array full of ones,
  - **empty**: the function empty creates an array whose initial content is random and depends on the state of the memory. By default, the dtype is float64.
  - **arrange**: the function arrange creates a range: from, to, step (the to is exclusive)

166

## Array creation III

```
x = np.zeros((3, 4))
print(x)

y = np.ones( (2,3,4), dtype=np.int16 )
print(y)

z = np.empty( (2,3) )
print(z)

a = np.arange(0.0, 1.01, 0.1)  # [0.0 0.1 0.2 ... 0.9 1.0]
print(a)
```

```
[[0.  0.  0.  0.]
 [0.  0.  0.  0.]
 [0.  0.  0.  0.]]
[[1  1  1  1]
 [1  1  1  1]
 [1  1  1  1]]
[[1.5  2.   3. ]
 [4.   5.   6.  ]
 [0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1. ]]
```

Output

167

## Random numbers

- Mit der Random Funktion können Array mit zufälligen Zahlen initialisiert werden:

```
x = np.random.random(3)  # Zahlen zwischen 0..1
print(x)

# Zahlen aus Normalverteilung
y = np.random.normal(loc=0.0, scale=1.0, size=10)
print(y)

# Zahlen aus Auswahl
z = np.random.choice([2,4,6,8], 10)
print(z)
```

```
[0.80217922 0.41181568 0.98317354]

[ 0.17643768  0.13406789  0.62499216 -0.99067793  0.72721655 -1.18774602
  0.03740755  1.23494155 -0.7377875   0.21212164]

[2  2  6  8  6  6  4  6  4  8]
```

Output

168

## Array slicing

```
x = np.array(
    [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]])

print(x[0, :])      # first line
print(x[:, 0])      # first column
print(x[0, 0])      # first element of first column
print(x[0:2, 1])    # first 2 elements of 2nd column
```

```
[1 2 3]
[4 5 6]
1
[2 5]
```

Output

169

## Onedimensional array slicing

```
a = np.arange(10)**3
print(a)      # a = [0 1 8 27 64 125 216 343 512 729]
print(a[2])   # 8
print(a[2:5]) # [8 27 64]

a[0:6:2] = 1000 # from position 0 to 6 exclusive, set every 2nd element to 1000
               # (equivalent to: a[:6:2] = 1000)
               # [1000, 1, 1000, 27, 1000, 125, 216, 343, 512, 729]

a[::-1]       # reversed a
               # [729, 512, 343, 216, 125, 1000, 27, 1000, 1, 1000]
```

```
[ 0  1  8 27 64 125 216 343 512 729]
8
[ 8 27 64]
```

Output

170

## Multidimensional array slicing

```
def myfunc(x,y):
    return 10*x + y

b = np.fromfunction(myfunc,(5,4),dtype=int) # Initialisierung: x = 0...4 mit jeweils y = 0..3

print(a)          # [ [ 0, 1, 2, 3],
                  # [10, 11, 12, 13],
                  # [20, 21, 22, 23],
                  # [30, 31, 32, 33],
                  # [40, 41, 42, 43] ]

print(b[2,3])      # 23

print(b[0:5, 1])   # each row in the second column of b (equivalent to: b[:,1])
                  # [ 1, 11, 21, 31, 41]

print(b[1:3, :])   # each column in the second and third row of b
                  # [ [10, 11, 12, 13],
                  #   [20, 21, 22, 23] ]
```

171

## Operationen per Element

```
# Vergleiche (element-wise)
a = np.array([1,2,3,4,5])

b = a > 3          # Vergleich "element-wise" erzeugt Array mit boolean
print(b)          # b = [False False False True True]

c = a[b]           # Boolean array als Maske
print(c)          # c = [4 5]

d = a[a <= 3]      # Boolean array als Maske
print(d)          # d = [1 2 3]
```

```
[False False False  True  True]
[4  5]
[1  2  3]
[25 35 45 55]
[ 4.  6.  8. 10.]
```

Output

172

## Operationen per Element II

```
# Operationen wie +, -, *, /]
a = np.array([20,30,40,50])

b = a + 5
print(b)

c = a / 5
print(c)

# Arithmetic operators on arrays apply elementwise.
a = np.array([20,30,40,50])
b = np.array([10,15,20,25])
c = a-b
print(c)
```

```
[25 35 45 55]
[ 4.  6.  8. 10.]
[10 15 20 25]
```

Output

173

## nan, any and logical functions

```
x = np.nan           # undefiniertes Element
y = np.isnan(x)      # Test für undefiniertes Element
print(y)             # y = True

a = np.array([1,2,np.nan,4,5])
b = np.isnan(a)      # Test "element-wise" für undefiniertes Modell
print(b)             # b = [False False True False False]

c = np.any(b)         # Prüft ob mindestens ein Eintrag True ist
print(c)             # c = True

d = np.logical_not(b) # logical functions (logical_and/not/or/xor)
print(d)
```

```
True
[False False  True False False]
True
[ True  True False  True  True]
```

Output

174

## Statistik

```
a = np.array([7, 9, 4, 12, 6, 4])  
  
print(np.min(a))  
print(np.max(a))  
print(np.mean(a))      # Mittelwert  
print(np.std(a))       # Standard Abweichung  
print(np.sort(a))
```

```
4  
12  
7.0  
2.8284271247461903  
[ 4  4  6  7  9 12]
```

Output

175

## Abschnitt III

### Pandas

176



## Pandas

- ▶ Das Pandas DataFrame ist ähnlich wie die data.frame Klasse in R
- ▶ 2-Dimensionale Daten mit Labels für Spalte/Zeile
- ▶ Implementiert als wrapper um numpy.ndarray
  - Operationen wie mit numpy arrays
  - Man kann Masken brauchen (für die Indizierung)
- ▶ Ein DataFrame wird via die entsprechenden Klasse instanziiert

177

## DataFrame

```
import numpy as np
import pandas as pd

col_names = ["A", "B", "C"]
row_names = ["First", "Second", "Third"]
data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
df = pd.DataFrame(data, row_names, col_names)
print(df)

print(df.shape)      # Tuple with number of rows and columns
print(df.shape[0])   # Number of rows
print(df.shape[1])   # Number of columns
```

	A	B	C
First	1	2	3
Second	4	5	6
Third	7	8	9

```
(3, 3)
3
3
```

Output

178

## Slicing

```
# DataFrame
print(df)
```

```
# Spalten selektieren
c1 = df["A"]
print(c1)
```

```
c2 = df.B
print(c2)
```

	A	B	C
First	1	2	3
Second	4	5	6
Third	7	8	9

Output

First	1
Second	4
Third	7

Name: A, dtype: int64

First	2
Second	5
Third	8

Name: B, dtype: int64

## Slicing II

```
# DataFrame
print(df)

# Zeilen selektieren
# Label slicing ist inklusive End Label
r1_r2 = df["First":"Second"]
print(r1_r2)

# Slicing via Index ist exklusive End Index
r2 = df[1:2]
print(r2)

# Das loc Attribut erlaubt Label
# basiertes Slicing von Kolone und Zeile
x = df.loc["First":"Second", "B":"C"]
print(x)
```

	A	B	C
First	1	2	3
Second	4	5	6
Third	7	8	9

Output

	A	B	C
First	1	2	3
Second	4	5	6

	A	B	C
Second	4	5	6

	B	C
First	2	3
Second	5	6

## apply

- ▶ Die apply Methode erlaubt eine Funktion auf jedem Element oder einer Serie von Elementen eines DataFrame anzuwenden:

```
sq = df.apply(np.sqrt)
print(sq)
```

```
min_per_col = df.apply(min)
print(min_per_col)
```

```
def mycalc(x):
    return 2*x
mc = df.apply(mycalc)
print(mc)
```

	A	B	C
First	1.000000	1.414214	1.732051
Second	2.000000	2.236068	2.449490
Third	2.645751	2.828427	3.000000

A	1
B	2
C	3

A	1
B	2
C	3

A	1
B	2
C	3

	A	B	C
First	2	4	6
Second	8	10	12
Third	14	16	18

Output

## groupby

- ▶ Mit der groupby Methode werden Daten nach einer Spalte gruppiert
- ▶ Man kann dann auf dem zurückgegebenen Objekt Funktionen pro Gruppe anwenden

```
col_names = ["A", "B"]
```

```
data = [[1, 2],
        [1, 4],
        [2, 4],
        [1, 2]]
```

```
df = pd.DataFrame(data, columns=col_names)
```

```
grouped = df.groupby("A") # 1: 2 4 2 : mean = 8/3 = 2.67
                        # 2: 4 : mean = 4/1 = 4
```

```
print(grouped.mean())
```

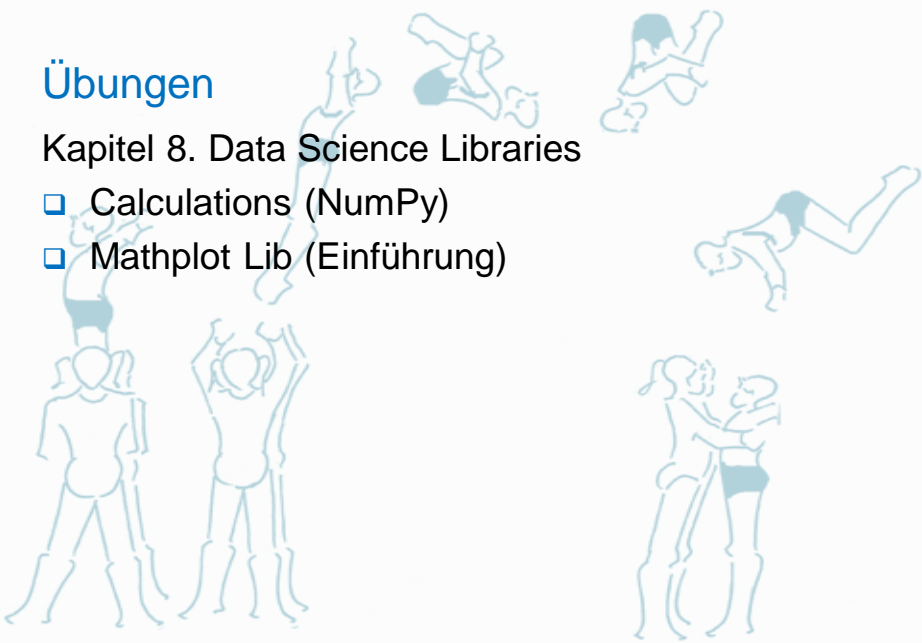
	B
A	
1	2.666667
2	4.000000

Output

# Übungen

## Kapitel 8. Data Science Libraries

- ▣ Calculations (NumPy)
- ▣ Mathplot Lib (Einführung)

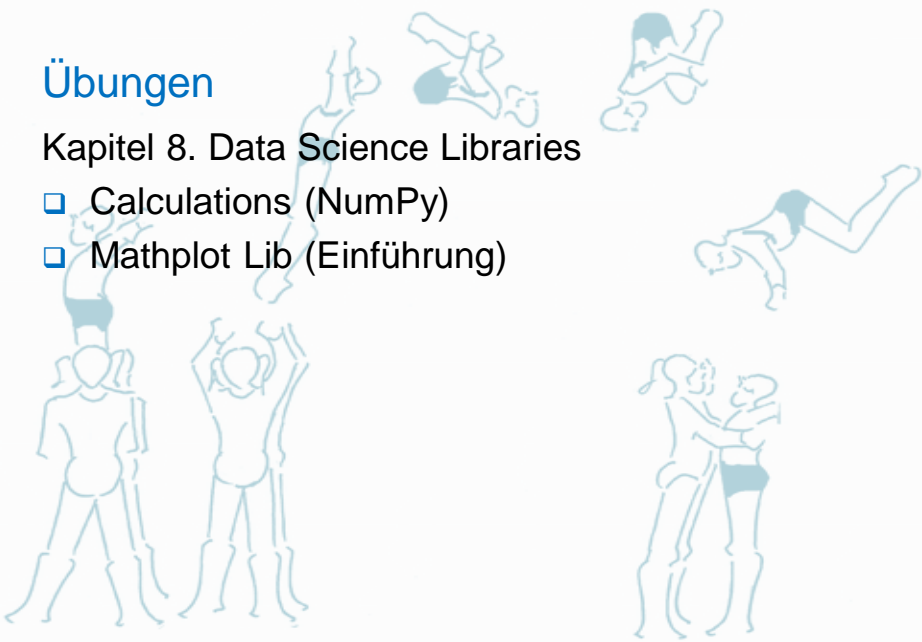


Copyright © iten-engineering.ch Python Einführung 183

183

## Kapitel 9

Anwendungsbeispiele



Copyright © iten-engineering.ch Python Einführung 184

184

# Übungen

## Kapitel 9. Anwendungsbeispiele

- ☐ SciKitLearn
- ☐ Bookservice
- ☐ Streamlit
  - ☐ HelloStreamlit
  - ☐ Sales


A light blue line-art illustration of several people in various acrobatic poses, including handstands and backflips, set against a light blue background.

Copyright © iten-engineering.ch Python Einführung 185

185

## Kapitel 10

# Weitere Übungen

A light blue line-art illustration of several people in various acrobatic poses, including handstands and backflips, set against a light blue background.

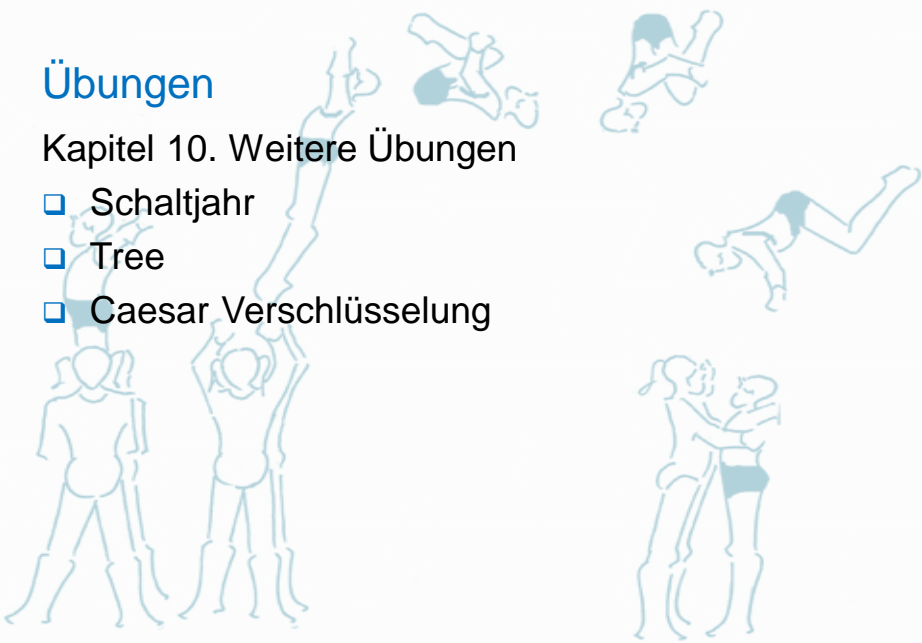
Copyright © iten-engineering.ch Python Einführung 186

186

## Übungen

### Kapitel 10. Weitere Übungen

- ❑ Schaltjahr
- ❑ Tree
- ❑ Caesar Verschlüsselung




Copyright © iten-engineering.ch Python Einführung 187

187

## Kapitel 11

# Literatur und Weblinks



Copyright © iten-engineering.ch Python Einführung 188

188

## Literatur und Weblinks

- ▶ Python Home  
<https://www.python.org>
- ▶ Python Docs  
<https://docs.python.org/3>
- ▶ NumPy  
<https://numpy.org>
- ▶ SciPy  
<https://www.scipy.org>
- ▶ Pandas  
<https://pandas.pydata.org>
- ▶ SciKit Learn  
<https://scikit-learn.org/stable>

189

## Kapitel 12

### Anhang

190

## Reserved Words

and	assert	break	class
continue	def	del	elif
else	except	exec	finally
for	from	global	if
import	in	is	lambda
not	or	pass	print
raise	return	try	While
with			

191

## Escape Sequence

Escape-sequence	Purpose
<code>\n</code>	New line
<code>\\</code>	Backslash character
<code>\'</code>	Apostrophe '
<code>\"</code>	Quotation mark "
<code>\a</code>	Sound signal
<code>\b</code>	Slaughter (backspace key symbol)
<code>\f</code>	The conversion of format
<code>\r</code>	Carriage return
<code>\t</code>	Horizontal tab
<code>\v</code>	Vertical tab
<code>\xhh</code>	Character with hex code hh
<code>\ooo</code>	Character with octal value ooo
<code>\0</code>	Character Null (not a string terminator)
<code>\N{id}</code>	Identifier ID of Unicode database
<code>\uhhhh</code>	16-bit Unicode character in hexadecimal format
<code>\Uhhhhhhhh</code>	32-bit Unicode character in hexadecimal format

192



## Argumente «entpacken»

- ▶ Argumente «entpacken» mit `*args` und `**kwargs`
  - `*list` entpackt Elemente einer Liste
  - `**dict` entpackt Elemente eines Dictionary

```
def f4(a, b, c=None, d=None):
    print("f4: a={}, b={}, c={}, d={}".format(a,b,c,d))

f4(*[1,2])                # Unpack elements from a list
f4(*[1,2], d=4)

f4(1, 2, **{"c":3, "d":4}) # Unpack elements from a dict
f4(1, 2, **{"d":4, "c":3})
```

```
f4: a=1, b=2, c=None, d=None
f4: a=1, b=2, c=None, d=4
f4: a=1, b=2, c=3, d=4
f4: a=1, b=2, c=3, d=4
```

Output

## Funktion mit variabler Anzahl Argumente

- ▶ Funktionsdefinition mit `*` und `**`:
  - Konventionell verwendet man `*args` und `**kwargs`
  - `args` enthält alle zusätzlichen «Positional» Argumente
  - `kwargs` enthält zusätzliche «Keyword» Argumente

```
def f5(a, *args, k=9, **kwargs):
    print("f5: a={}, args={}, k={}, kwargs={}".format(a,args,k,kwargs))

f5(1)
f5(1,2,4,6,k=7,x=9,y=11)
```

```
f5: a=1, args=(), k=9, kwargs={}
f5: a=1, args=(2, 4, 6), k=7, kwargs={'x': 9, 'y': 11}
```

Output

## Multiple return values

- ▶ Python functions can return multiple variables.
  - These variables can be stored in variables directly.
  - A function is not required to return a variable, it can return zero, one, two or more variables.
- ▶ This is a unique property of Python, other programming languages such as C++ or Java do not support this.

```
def getPerson():
    name = "Leona"
    age = 35
    country = "UK"
    return name,age,country

name,age,country = getPerson()
print(name)
print(age)
print(country)
```

Quelle: <https://pythonbasics.org/multiple-return>

## Built-in Functions

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Quelle: <https://docs.python.org/3.3/library/functions.html>

## Built-in Functions - zip

- Make an iterator that aggregates elements from each of the iterables

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> list(zipped)
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*zip(x, y))
>>> x == list(x2) and y == list(y2)
True
```

Quelle: <https://docs.python.org/3.3/library/functions.html#zip>

## Frozen Sets

- The `frozenset()` function returns an immutable `frozenset` initialized with elements from the given iterable.
- If no parameters are passed, it returns an empty `frozenset`.

```
# tuple of vowels
vowels = ('a', 'e', 'i', 'o', 'u')

fSet = frozenset(vowels)
print('The frozen set is:', fSet)
print('The empty frozen set is:', frozenset())

# frozensets are immutable
fSet.add('v')
```

### Output

```
The frozen set is: frozenset({'a', 'o', 'u', 'i', 'e'})
The empty frozen set is: frozenset()
Traceback (most recent call last):
  File "<string>, line 8, in <module>
    fSet.add('v')
AttributeError: 'frozenset' object has no attribute 'add'
```

Quelle: <https://www.programiz.com/python-programming/methods/built-in/frozenset>

## Shallow and deep copy

- ▶ The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances):
  - A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.
  - A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

Quelle: <https://docs.python.org/3/library/copy.html>

Copyright © iten-engineering.ch

Python Einführung

199

199

## Shallow and deep copy II

- ▶ Two **problems** often exist with deep copy operations that don't exist with shallow copy operations:
  - Recursive objects (compound objects that, directly or indirectly, contain a reference to themselves) may cause a recursive loop.
  - Because deep copy copies everything it may copy too much, such as data which is intended to be shared between copies.
- ▶ The **deepcopy()** function avoids these problems by:
  - keeping a memo dictionary of objects already copied during the current copying pass; and
  - letting user-defined classes override the copying operation or the set of components copied.

Quelle: <https://docs.python.org/3/library/copy.html>

Copyright © iten-engineering.ch

Python Einführung

200

200

## Shallow and deep copy III

```
import copy

print("# Copy with: =")
old = [[1, 2, 3], [4, 5, 6], [7, 8, 'a']]
new = old
new[2][2] = 9
print('Old list:', old, "with ID", id(old) )
print('New list:', old, "with ID", id(new) )

print("# Copy with: copy")
old = [[1, 2, 3], [4, 5, 6], [7, 8, 'a']]
```

```
new = copy.copy(old)
new[2][2] = 9
print('Old list:', old, "with ID", id(old) )
print('New list:', old, "with ID", id(new) )

print("# Copy with: deepcopy")
old = [[1, 2, 3], [4, 5, 6], [7, 8, 'a']]
new = copy.deepcopy(old)
new[2][2] = 9
print('Old list:', old, "with ID", id(old) )
print('New list:', old, "with ID", id(new) )
```

```
# Copy with: =
Old list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] with ID 2204789025864
New list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] with ID 2204789025864

# Copy with: copy
Old list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] with ID 2204821800712
New list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] with ID 2204838633032

# Copy with: deepcopy
Old list: [[1, 2, 3], [4, 5, 6], [7, 8, 'a']] with ID 2204837090376
New list: [[1, 2, 3], [4, 5, 6], [7, 8, 'a']] with ID 2204821800712
```

Output

## Operator Overloading

Operator	Method
+	__add__(self, other)
-	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
%	__mod__(self, other)
<	__lt__(self, other)
<=< code>	__le__(self, other)
==	__eq__(self, other)
!=	__ne__(self, other)
>	__gt__(self, other)
>=	__ge__(self, other)

Quelle: <https://stackabuse.com/overloading-functions-and-operators-in-python>

## Python Debugger

- ▶ Das Python Debugger Modul heisst `pdb` und stellt einen interaktiven Source Code Debugger zur Verfügung.
- ▶ Man kann Breakpoints setzen, Schritt für Schritt den Code ausführen, die Stack Frames inspizieren und mehr.
- ▶ In einer integrierten Entwicklungsumgebung kann man aber in gewohnter Weise Debuggen und muss sich nicht um das `pdb` Modul kümmern